Approved for public release; distribution is unlimited.

PUBLIC–KEY CRYPTOGRAPHY: A HARDWARE IMPLEMENTATION AND NOVEL NEURAL NETWORK–BASED APPROACH

by

Phong Nguyen
Lieutenant, United States Navy
B.S.E.E., United States Naval Academy, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

September, 1992

Y CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

PORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
CURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
CLASSIFICATION/DOWNGRADING SCHEDULE	
RFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)

ME OF PERFORMING ORGANIZATION rical and Computer Eng. l Postgraduate School	6b. OFFICE SYMBOL (if applicable) Code 32	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
DRESS (City, State, and ZIP Code) terey, CA 93943-5000		7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
ME OF FUNDING/SPONSORING GANIZATION	8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
DRESS (City, State, and ZIP Code)		10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION N

TLE (Include Security Classification) PUBLIC-KEY CRYPTOGRAPHY: A HARDWARE IMPLEMENTATION AND NOVEL NEURAL NETW ORK-BASED APPROACH (U)

RSONAL AUTHOR(S)
ng Nguyen

YPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
er's Thesis	FROM 03/90 TO 09/92	1992 September	108

PPLEMENTARY NOTATIOThe views expressed in this thesis are those of the author and do not reflect the official y or position of the Department of Defense or the United States Government.

COSATI CODES		
LD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Cryptography, Public-Key, Secret-Key, Discrete Logarithm, F Exponentiation, Diffie-Hellman, RSA, Inverse, GCD, Neural Netwo Back-Propagation, Factorization, Sum of Residues, Modulo Reduction

STRACT (Continue on reverse if necessary and identify by block number)

ne concealment of information passed over a non-secure communication link lies in the complex field of cryp phy. Furthermore, when absolutely no secure channel exists for the exchange of a secret key with which data pted and decrypted, the remedy lies in a branch of cryptography known as public-key cryptosystem (PKS). Thi provides an in-depth study of the public-key cryptosystem. Essential background knowledge is covered leadin a VLSI implementation of a fast modulo exponentiator based on the sum of residues (SOR) method. Fast modul nentiation is vital in the most popular PKS schemes. Furthermore, since all cryptosystems make use of som of mapping functions, a neural network - an excellent non-linear mapping technique - provides a viable mediu which a possible cryptosystem can be based. In examining this possibility, this thesis presents an adaptation ck-propagation neural network to a "pseudo" public-key arrangement. Following examinations of the networ management system is then devised. Finally, a complete top-down block diagram of an entire cryptosyste on the neural network of this study is proposed.

STRIBUTION/AVAILABILITY OF ABSTRACT NCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
AME OF RESPONSIBLE INDIVIDUAL an Yang	22b. TELEPHONE (Include Area Code) (408) 646-2266	22c. OFFICE SYMBOL EC/Ya

RM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

ABSTRACT

The concealment of information passed over a non–secure communication link lies in the complex field of cryptography. Furthermore, when absolutely no secure channel exists for the exchange of a secret key with which data is encrypted and decrypted, the remedy lies in a branch of cryptography known as public–key cryptosystem (PKS). This thesis provides an in–depth study of the public–key cryptosystem. Essential background knowledge is covered leading up to a VLSI implementation of a fast modulo exponentiator based on the sum of residues (SOR) method. Fast modulo exponentiation is vital in the most popular PKS schemes. Furthermore, since all cryptosystems make use of some form of mapping functions, a neural network – an excellent non–linear mapping technique – provides a viable medium upon which a possible cryptosystem can be based. In examining this possibility, this thesis presents an adaptation of the back–propagation neural network to a "pseudo" public–key arrangement. Following examinations of the network, a key management system is then devised. Finally, a complete top–down block diagram of an entire cryptosystem based on the neural network of this study is proposed.

ACKNOWLEDGMENTS

The author extends his gratitude to Professor Yang for the insights and encouragement throughout the research. In addition, credit must be aknowledged for LT Daniel Sullivan's assistance in the neural network section and for LT Arthur Billingsley's general support in this thesis. Finally, my appreciation and admiration are extended to the faculty and staff of the Electrical and Computer Engineering Department for the outstanding guidance and opportunity provided throughout the past two years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

In the recent past, there possibly was a time when protection of vital electronic information was not considered a necessity and therefore not deemed to be a topic of common interest. Such a time is forever behind us. In our time, information is most often passed across a public telecommunication medium. Whether this medium be a telephone line or satellite link, there exist eavesdropping methods which are so sophisticated and efficient that no information is physically secure. How then is one to revert to the inherent privacy of the past? The answer to this question and thus the solution to concealment of information lie in the complex science of cryptography.

Cryptography is the field involving the preparation of messages intended to be incomprehensible to all except those who legitimately possess the means to recover the original information [Ref 1]. At present, the fastest and most popular cryptosystems employ some convention of mapping a set of numbers representing data to another set of numbers (encryption). The recovery of data is done by simply reversing the mapping process so as to obtain the original content (decryption). Often, this type of mapping is governed by the notion of a key. In order to provide the essential element of secrecy, system users must provide this key which is normally a privately or semi–privately known string of characters or bits. For a cryptosystem to be completely secure, knowledge of both the mapping function and key is required to recover the original text from encrypted text.

Of the cryptosystems which use the forementioned concept of a key, two distinct categories are made: secret–key and public–key.

As suggested by the name, a cryptosystem is *secret–key* if the key must be secretly agreed upon prior to any parties being able to communicate through the system. In this arrangement, both parties normally have the same key which is used in both encryption and decryption. Algorithms implementing this scheme are labeled symmetric. Intuitively, one recognizes a severe restriction in the secret-key system: an advance agreement on the key over a *secure* channel. When such a channel is not readily available, the topic of this thesis, *public-key* cryptosystem (PKS), is the remedy.

Most PKS systems use an asymmetric algorithm whereupon separate keys are required for encryption and decryption. This scheme allows the passing of keys, most likely encryption keys, over an unsecure channel without any compromise to the system's safety. In boasting this versatile capability, however, public-key system must pay a price, namely a reduction in system speed [Ref 2]. Currently, PKS is much slower than secret-key, too slow for large quantities of data. For this reason, its use is often limited to the exchange of keys in secret-key systems. In the future, along with advancements in technology, perhaps this speed barrier will be lifted yielding more opportunity for the employment of PKS.

It is in the spirit of this future that this thesis is presented. It is an in-depth study of the public-key cryptosystem. First, the mathematical basis behind PKS is covered so as to establish an essential background knowledge in a somewhat esoteric subject. Second, the capability of VLSI implementation of PKS is explored via a fast modulo exponentiator, a hardware device required in two of the most popular public-key systems. A vital component of the fast modulo exponentiator, a modulo reduction unit, is designed with MAGIC tools [Ref 3], validated with RNL simulation [Ref 4], and examined for possible use. Finally, to conclude the scope of this research, a completely novel approach to PKS is proposed: a possible implementation of neural networks in public-key cryptography.

II. MATHEMATICAL BASIS FOR THE DEVELOPMENT OF PUBLIC–KEY CRYPTOSYSTEMS

Compared to the complexity of conventional engineering mathematics, the concepts behind the algorithms for public–key cryptosystem are elementary in nature yet without complete understanding of them, no initial familiarization to the system is possible. Due to this realization, this chapter concentrates heavily on the mathematics of asymmetric cryptography. It provides a basic overview of modulo arithmetic, fast exponentiation, and discrete logarithm. It also outlines a background knowledge in artificial neural networks, a branch of engineering upon which a completely new angle in cryptography is based. Furthermore, the fundamentals of public–key cryptosystems are covered using two well–established examples, the Diffie–Hellman and RSA systems. Finally, the chapter concludes with the problem of cryptoanalysis: the purpose of all cryptosystems.

A. MODULO ARITHMETIC

Modulo arithmetic is a branch of integer mathematic best explained by an example.

Simply,

$$21 \equiv 3 (\bmod 9)$$

or

$$21 = 3 + 9 \times 2.$$

This operation is commonly described as 21 divided by 9 equals 2 with remainder of 3.

When written as $x \equiv y(\text{mod } z)$, by convention x is said to be "congruent to y modulo z." Congruency applies if and only if

$$x = y + k \times z$$

where k is any integer. Also y is called a residue mod z of x if and only if $x = y(\text{mod } z)$.

Note that $-15(\text{mod } 6) \equiv -3(\text{mod } 6)$.

Clearly, for any z, y belongs to a complete set of residues $\{0, 1, 2..., z-1\}$. From this complete set of residues, there exists a subset called a reduced set of residues which has elements relatively prime to the modulus z. For example, a complete set of residues modulo 12 is $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$. From this, only $\{1, 5, 7, 11\}$ does not have a common factor with 12 (0 excluded); it is therefore a reduced set [Ref 2].

For a modulo prime, clearly the reduced set of residues contains all elements of the complete set except for 0. Therefore for a prime n, the reduced set of residues has $(n-1)$ elements. In addition, generally the reduced set of residues for a product of two primes m and n has $((m-1)(n-1))$ elements and that for a prime power n^r has $(n-1)n^{(r-1)}$ elements. Commonly, the number of elements in a reduced set of residues for modulo n is referred to as the Euler Totient function $\phi(n)$ [Ref 2]. Table 2.1 shows $\phi(n)$ for several n [Ref 2].

Like normal integer arithmetic, addition and multiplication in integer modulo n abide by the laws of associativity, commutativity and distributivity [Ref 2].

Theorem 1 [Ref 2]:

$$(a + b)(\text{mod } n) = (a \text{ mod } n + b \text{ mod } n) \text{ mod } n$$

n	Reduced set	$\phi(n)$
n prime	$1, 2, ..., n-1$	$n-1$
n^2 (n prime)	$[1, 2, ..., n-1, n+1,$ $..., 2n-1, 2n+1,$ $..., n^2-1]$	$n(n-1)$
.	.	.
.	.	.
.	.	.
n^r(n prime)	$[1, 2, ..., n^r-1$ ...multiples of $n < n^r]$	$(n^r-1)-(n^{r-1}-1)$ $= n^{r-1}(n-1)$
pq(p, q primes)	$[1, 2, ..., pq-1$ ...multiples of p ...multiples of $q]$	$(pq-1)-(q-1)-(p-1)$ $= (p-1)(q-1)$
.	.	.
.	.	.
.	.	.
$\prod_{i=1}^{t} p_i^{e^i}$; ($p^i$ primes)		$\prod_{i=1}^{t} p_i^{e^i-1}(p_i-1)$

TABLE 2.1: EULER'S TOTIENT FUNCTIONS

Theorem 2 [Ref 2]:

$$ab(\text{mod } n) = (a \text{ mod } n \times b \text{ mod } n) \text{ mod } n$$

These two theorems form the basis for the development of fast modulo exponentiation.

B. FAST MODULO EXPONENTIATION

Many public–key cryptosystem requires the computation of x^k mod n, with n and k being extremely large numbers (in excess of 256 bits.) A naive solution would be to multiply by x a repetition of $k-1$ times then taking the modulo of the large result. At best, this is both cumbersome and inefficient for today's computers due to finite word length limit. Fortunately, there is an algorithm which avoids this

Iteration(i)	k bit	square ops $\times pp_{i-1}$	pp_i
1	0	5^1 but kbit= 0 so no op	**1** (remains the same)
2	1	$5^2 \times \mathbf{1}$	5^2
3	0	$(5^2)^2$ but kbit= 0 so no op	$\mathbf{5^2}$ (remains the same)
4	1	$((5^2)^2)^2 \times \mathbf{5^2}$	5^{10}

TABLE 2.2: EXAMPLE FAST EXPONENTIATION FOR 5^{10}

straightforward method: fast modular exponentiation [Ref 5].

Taking advantage of Theorem 2, the exponentiation is faster when performed by repeated squaring operations coupled with conditional multiplication by the partial product according to the binary representation of the exponent. This is best explained by an example.

Example:

Suppose we are required to find 5^{10} mod 9.

let $x = 5; k = 10; m = 9$

Using $pp_0 = 1$ and

$$pp_i = \begin{cases} x^{2^{i-1}} \times pp_{i-1} & \text{if } k_i = 1 \\ pp_{i-1} & \text{if } k_i = 0 \end{cases}$$

k in binary is 1010. In accordance to k, bit by bit from least significant bit (LSB) first, the squaring of x occurs iteratively for every k bit (0 or 1) but the result is multiplied by the partial product only when k bit is 1. All the while, modulo operation is performed in each squaring or multiplication in order to maintain a manageable intermediate result. The partial product is always initialized to 1 (partial product at iteration step 0, $pp_0 = 1$). Let's examine Table 2.2 for clarity. From the result of Table 2.2, indeed we have accomplished 5^{10}. □

If we incorporate the modulo operation into each iteration according to Theorem 2, the modulo problem is also solved. Table 2.3 incorporates modulo reduction to

Iteration	k bit	Square ops	Multiply ops	pp_i
1	0	(5^1) mod 9 = **5**		*1*(Init)
2	1	$(\mathbf{5}^2)$ mod 9 = *7*×	*1* mod 9	= **7**
3	0	(7^2) mod 9 = *4*		
4	1	(4^2) mod 9 = *7*×	**7** mod 9	= 49

TABLE 2.3: EXAMPLE FAST EXPONENTIATION AND MODULO OF 5^{10} mod 9

the previous example.

Example:

5^{10} mod 9

Table 2.3 outlines in detail the process until a partial product of 49 is obtained. Note that the result of the square operation becomes the number to be squared in the next iteration. Also the previous partial product is the number in the multiplying operation if the k bit is 1. In this example, since 49 mod 9 $=$ 4, indeed 5^{10} mod 9 (which also equals 4) is performed. □

In this example the savings in multiplications is 4 (5 versus 9 using the naive method). For larger number applications, let a be the number of binary bits of the exponent k and b be $\log_2 a$. Using fast exponentiation, the number of multiplications (call it X) is bounded by $b + 1 < X < 2b + 1$ depending on the number of 1's and 0's in k. X with fast exponentiation grows linearly in length of k and is considerably smaller then X obtained by the straightforward method of multiplying by $k - 1$ times [Ref 5].

Appendix A contains a C program implementing fast modular exponentiation using the above algorithm. It should be noted that the program is not suitable for numbers exceeding the capability of the computer. Most computers have 32 bits resolution therefore results which are greater than 32 bits are likely to be too large. This limitation, however, is resolved by using hardware for fast modular exponentiation as will be shown in Chapter III.

C. DISCRETE LOGARITHM

Discrete logarithm is the branch of mathematics centered on the solution to the exponent of a powered number; namely, finding x in $a^x \equiv b \bmod n$ when given a, b, n.

Example:

$a = 3; b = 4; n = 11;$

$$3^1 \bmod 11 = 3$$

$$3^2 \bmod 11 = 9$$

$$3^3 \bmod 11 = 5$$

$$3^4 \bmod 11 = 4$$

so $x = 4$.

Given a large modulus n and a, b (greater than 100 digits magnitude), discrete logarithm is classified as a non–deterministic polynomials problem; the solution to which is extremely difficult and impractical to derive [Ref 6]. Therefore its use is prevalent throughout many public–key cryptosystems.

D. INVERSES

Unlike integer arithmetic, modulo arithmetic often has inverses. Given $a \in \{0, n-1\}$, there could be a unique $b \in \{0, n-1\}$ such that

$$ab(\bmod n) \equiv 1 \; [Ref\ 2]$$

A systematic method to compute inverses involves the notion of the greatest common divisor (gcd). Conventionally, $gcd(a, b)$ is an integer c such that a/c and

b/c result in the smallest possible integer value. For example, $gcd(8,12) = 4$ but $gcd(8,16) = 8$.

From the mathematics of gcd, we pose:

Lemma 1 [Ref 2]: if $gcd(a,n) = 1$ then

$$a_i \bmod n \neq a_j \bmod n; 0 \leq i,j \leq n$$

Fermat's Theorem [Ref 2]: p is a prime and $gcd(a,p) = 1$ then

$$a^{(p-1)}(\bmod\ p) = 1$$

Theorem 3 [Ref 2]:if $gcd(a,n) = 1$ then an $a^{-1}, 0 < a^{-1} < n$ exists such that

$$aa^{-1} \equiv 1(\bmod\ n)$$

Theorem 4 [Ref 2]: if $gcd(a,n) = 1$ then

$$a^{\phi(n)} \bmod n = 1$$

Recall $\phi(n)$ is the number of elements in a reduced set of residues (Table 2.1).

From the above Theorems, Euclid's algorithm is developed to find $gcd(a,n)$ as well as inverse $a^{-1}(\bmod\ n)$ of $a \bmod n$. It is not within the scope of this study to detail the foundation of this algorithm. If further information is preferred, reference 2 is suggested for consultation. For the purpose of this thesis, C programs for gcd and inverse are provided in Appendix A [Ref 2].

E. ARTIFICIAL NEURAL NETWORK

In 1985, Ackley, Hinton and Sejnowski [Ref 7] applied a back-propagation neural network to encode orthogonal binary vectors of length N using $log_2 N$ hidden units. Following this, Cottrell, Munro and Zipser [Ref 8] used the same type of network to achieve image (data) compression. Both these two application examples involved a special form of mapping via neural networks and, thus, suggested a possible use in cryptography. In fact, they are inspirational for the work of Chapter IV in this thesis which explores in detail the possibility of implementing neural networks in a novel public-key cryptosystem. In light of this, this section provides a basic understanding of neural networks, especially the back-propagation neural network.

A formal definition of a neural network is:

"A neural network is a parallel, distributed information processing structure consisting of processing elements (which can possess a local memory and can carry out localized information processing operations) interconnected via unidirectional signal channels called connections. Each processing element has a single output connection that branches into as many collateral connections as desired; each carries the same signal- the processing element ouput. This ouput signal can be of any mathematical types. The information of each element can be arbitrary with the restriction that it must be completely local; it must depend only on the current values of arriving input signals at and on values in local memory." [Ref 9]

Having defined a neural network, the basic unit, a processing element, is shown in Figure 2.1. The processing element has many input connections combined by a simple summation. The combination is then transformed through a transfer function. The function of interest here is a hyperbolic tangent. The single ouput of the element is fanned out to several ouput paths which then become inputs of other elements. The ouput to input connections each has a corresponding weight. Since the connections prior to entering the elements are modified by the weights, the summation within each element is a weighted sum. The actual mathematical process within an element is thus:

$$f(\sum_i w_{ij}x_i); \quad i = layer; \; j = number\ of\ weights$$

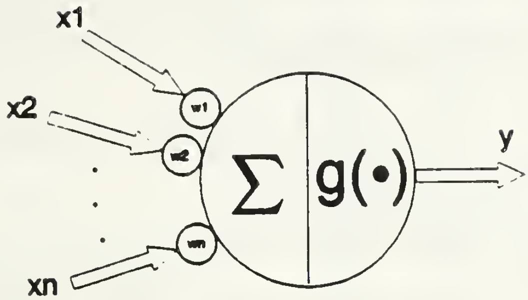

Figure 2.1: A Processing Element

An overall neural network consists of many processing elements joined together as previously discussed. A typical neural network, a back-propagation network in this case, is shown in Figure 2.2 [Ref 10]. For organization purpose, processing elements are grouped into layers. A normal network is composed of two layers with connections to the outside world: an input buffer where data is entered and an output buffer where the response of the network to the given input is stored. Layers between the input and ouput layers are named hidden layers [Ref 10].

There are currently many types of neural networks designed for multitude of applications. For the purpose of encoding and decoding in a cryptosystem where the mapping of input to output is almost always non-linear, a most suitable network is the back-propagation type.

A back-propagation neural network is a 3 to 5 layer network that behaves as an interpolative-associative mapping scheme. That is it has the ability to learn mapping by generalizing input/ouput pairs relationship [Ref 9]. Moreover, the network employs a supervised, delta-rule learning scheme whereupon the input stimulus and corresponding output are first presented to the system which in turn reduces the error between the actual output of each element and the desired ouput and gradually

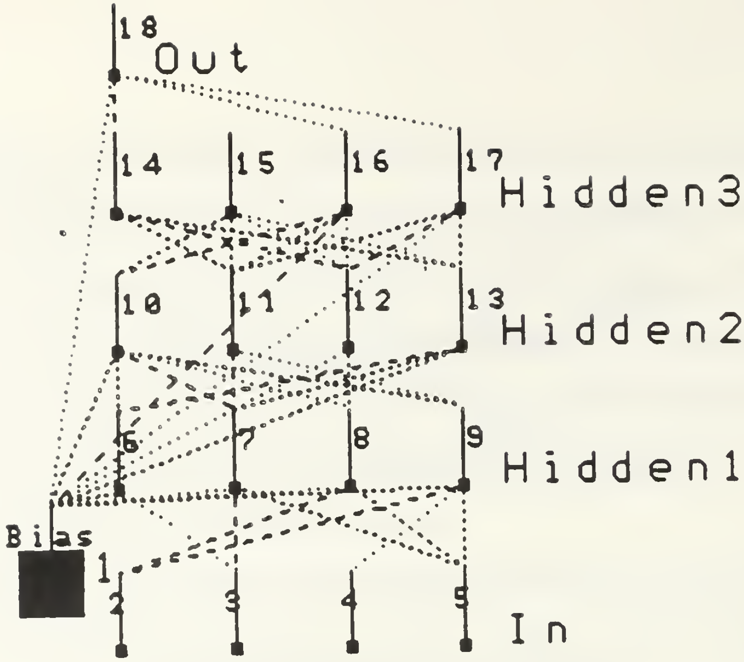

Figure 2.2: A Back–Propagation Network [Ref 10]

configures its weights to achieve the desired input/ouput mapping. After learning is accomplished, the error is reduced to minimum and the actual outputs of all inputs of interest will be approximately equaled to the theoretical output [Ref 10].

Having covered the necessary basics, the mathematical background for the back-propagation network is now provided. In order to establish a common convention, the notations used for this development is as follows.

- $x_j^{[s]} \equiv$ current output of j^{th} neuron in layer s,
- $w_{ji}^{[s]} \equiv$ connection weights joining i_{th} neuron in layer [s-1] to j^{th} neuron in layer s,
- $I_j^{[s]} \equiv$ weight summation of inputs to j^{th} neuron in layer s.

The mathematical process for single back-propagation element is:

$$x_j^{[s]} = f[\sum_i (w_{ji}^{[s]} x_i^{[s-1]})] = f(I_j^{[s]})$$

Given that the network has some global error function E, the critical parameter that is fed back through the layers is defined as:

$$e_j^{[s]} = -\partial E / \partial I_j^{[s]}$$

where $e_j^{[s]}$ is the local error of processing element j in layer s. Furthermore, using the chain rule twice yields:

$$e_j^{[s]} = f'(I_j^{[s]}) \sum_k (e_k^{[s+1]} w_{kj}^{[s+1]}).$$

The main mechanism in the back–propagation network is to forward the input to the output, determine the error at the output, then propagate the errors back using the above equations. Given knowledge of local errors, the final aim is to minimize the global error by modifying the weights.

This is done by using the gradient rule which dictates that the weights change in the direction of minimum error.

$$\triangle w_{ji}^{[s]} = -k(\partial E / \partial w_{ji}^{[s]})$$

where k is a learning coefficient.

Again using the chain rule:

$$\partial E / \partial w_{ji}^{[s]} = (\partial E / \partial I_j^{[s]})(\partial I_j^{[s]} / \partial w_{ji}^{[s]}) = -e_j^{[s]} x_i^{[s-1]}$$

$$\triangle w_{ji}^{[s]} = k e_{[s]} x_i^{[s-1]}.$$

For an in–depth derivation of all forementioned equations, the reader is referred to references 9 and 10.

Using the above equations in several iterations, an algorithm for the back–propagation network can be developed to train the network weights in converging to a given set of training data: inputs and corresponding outputs. It is not within the scope of this research to derive or show the algorithm; however, such an algorithm can be found in reference 9. In Chapter IV, a specific software package, Neuralware, will be utilize to set up a back-propagation network. The network will train with specific mapping functions so as to accomplish an encryption and decryption scheme in a newly-proposed "pseudo" public-key cryptosystem.

This concludes the necessary background in mathematic. We are now equipped with enough knowledge to explore the core of the public-key cryptosystem.

F. THE PUBLIC-KEY CRYPTOSYSTEM

The single foundation upon which all asymmetric cryptosystems are built is that of the one-way function. Such a function is practical to solve in one direction but within a range it is computationally infeasible for any algorithm to invert the solution taken over a range of elements [Ref 11]. A formal definition of a one-way function is beyond the scope of this study. An informal definition is that a one-way function is one in which for $f : x \rightarrow y$, it is easy to find $y = f(x)$ given x. However, given y, it is difficult to compute x such that $f(x) = y$ [Ref 12]. For use in cryptography, the difficulty must be great enough so as to render the solution impractical.

Currently we have a few one-way functions which are utilized exclusively in the public-key system. A good example of a one-way function is integer multiplication. Whereas the multiplication of large integers is relatively easy with current technology, the factoring of a large integer is time-consuming to the point of infeasibility. Another important example is modular exponentiation with large exponents. As previously discussed, fast exponentiation techniques makes the exponentiation practical. However, even with the best current algorithms and technology, the solution of a discrete logarithmic problem of such magnitude remains unattainable within a reasonable time [Ref 13]. To see how the two suggested one–way functions are used in public–key cryptosystems, in–depth studies of two systems are now provided: the Diffie–Hellman and RSA cryptosystems.

1. The Diffie–Hellman Scheme for Public–Key Cryptosystem

The first system to achieve the notoriety of a true public-key system was proposed by Diffie and Hellman seminal paper in 1976 [Ref 14]. It is in this paper that the discrete logarithm problem was first proposed as a candidate for a one–way function. The scheme is best summarized as follows.

Let n be a large integer and g, another integer, such that $g \in \{1, n-1\}$. Parties A and B establish n and g over insecure channels. A then chooses a large integer x and computes $g^x \bmod n$ while B chooses y and computes $g^y \bmod n$. Next, A and B exchanges their perspective computations again over insecure channels without divulging x and y. At this point A has g^y and n (possibly compromised over unsecured channels) and x which was never communicated to anyone. Similarly, B has g^x, n and y. A and B can construct the key as follows.

for A: $key = (g^y)^x \bmod n$

for B: $key = (g^x)^y \bmod n$

$$(g^y)^x \bmod n = (g^x)^y \bmod n$$

Clearly A and B now have the same key $(g^x)^y \bmod n$ which can be used for any cryptography systems. Because the operation of exponentiation with large exponent is slow, Diffie–Hellman is proposed only to make keys for faster private–key system such as DES so that the key will not be compromised [Ref 12].

Even if a cryptanalyst was able to intercept the exchanges for g, n, $g^x \bmod n$ and $g^y \bmod n$, he faces the problem of finding x and y from his known data. He must

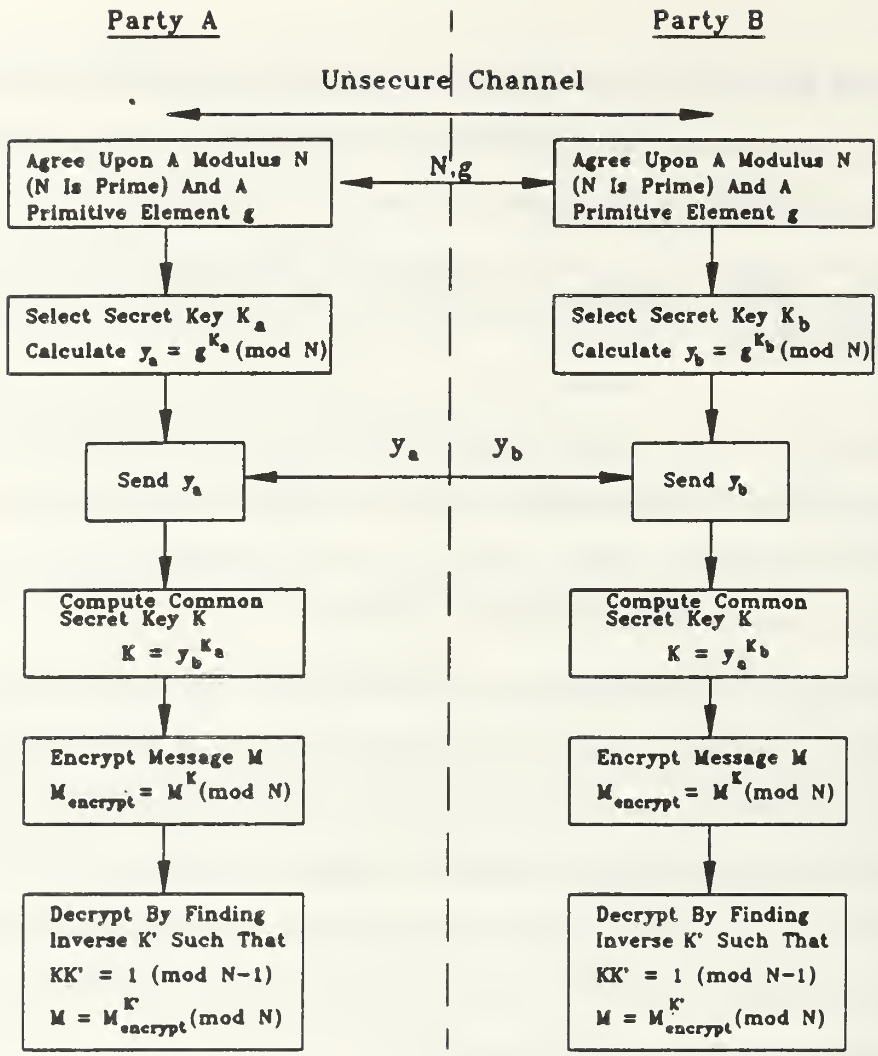

Figure 2.3: Block Diagram of Diffie–Hellman Cryptosystem

solve a discrete logarithm problem, an NP class problem, which, to date, is accepted to be infeasible within certain time restraints [Ref 13]. A summarizing block diagram of the Diffie–Hellman cryptosystem is provided in Figure 2.3. Moreover, an example of its application is hereby offered.

Example [Ref 13]:

Let $g = 7$ and $n = 2 \times 739(7^{149} - 1)/6 + 1$.

Party A chooses a secret x, compute and send 7^x to B.

B receives 7^x=

12740218011997394688242692443343228497493820425869316216545577352903 22
9146790959986818609788130465951664554581442805880767660337 81

Party B chooses a secret y, compute and send 7^y to A.

A receives 7^y=

1801622852874531024447828348348367998950159670466953466973130251217 34
0599537720584759581176910625380692101651848662362137934026803049

Now both A and B can compute 7^{xy} and mod it with n to establish secret key 7^{xy} mod n. Since a party other than A and B does not know either x or y in this case, it is infeasible to attempt finding 7^{xy}.

Note: The numbers in this example are obtained from reference 13 where neither x nor y was divulged. This author has been unable to find their values. In the original article, a challenge of 100 dollars was offered to anyone who could solve for x and y and thus 7^{xy}.□

Presently, the Diffie–Hellman scheme remains trustworthy because the discrete logarithm problem is still a difficult one to solve. Nevertheless, no one has proven beyond a doubt that it is impossible to solve. In fact, many algorithms do exist which can derive the solution. The only setback is that even the best of them is not fast enough with current technology. For more safety, the integers x and y can simply be increased in magnitude and for the worst case, an establishment of new key within an acceptable time interval can render any cryptoanalysis harmless.

2. The RSA Cryptosystem

Invented in 1978, the Rivest, Shamir and Adleman (RSA) public-key cryptosystem incorporates two one-way functions: the discrete logarithm and factorization problems. The security guaranteed by this system is so sound that since its inception until present, it has been accepted as the most popular method of public-key encryption [Ref 15]. The elegance and subtle power of the RSA system is summarized as follows.

Party A generates 2 random primes of approximately 130 bits each, p and q. The product pq is then computed and called n. The number of reduced residues elements is next obtained: $\phi(n) = (p-1)(q-1)$ (see Table 2.1). In turn, an integer e is generated such that $gcd(e, \phi(n)) = 1$. A now has the public key $< e, n >$ which can be published to B through insecured channels.

Having the public key, party B can encrypt a message by transforming the message into an integer value m. m is then encrypt by:

$$Encryp(m) = m^e \bmod n$$

In order to be able to decipher $Encryp(m)$, A must make a private key from $\phi(n)$ and e. Such a key, D, is found by using Euclid's algorithm (Appendix A) so that,

$$De = 1 \bmod \phi(n)$$

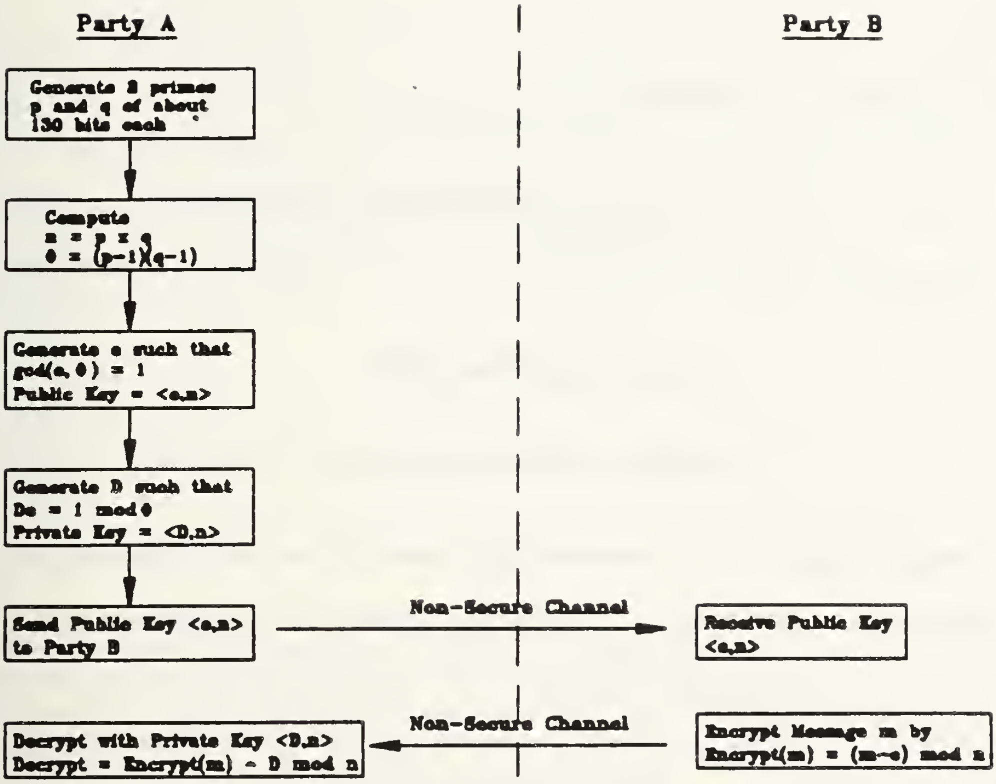

Figure 2.4: Block Diagram of RSA Cryptosystem

Once D is found, the deciphering is simply done by,

$$Deciph(Encryp(m)) = (Encryp(m))^D \bmod n$$

Proof [Ref 6]:

Given all parameters above, by Euler's Theorem:

if $De \equiv 1 \bmod (\phi) \rightarrow m^{De} \equiv m \bmod n$

$$\rightarrow m^{De} \bmod n = m$$

Figure 2.4 clarifies the process. In addition, a pedagogical example of RSA at work is shown below.

Example:

(Use actual Appendix A programs)

Let $p = 7; q = 13 \rightarrow n = 7 \times 13 = 91; \phi(n) = (7-1)(13-1) = 72$

Pick $e = 5$ and $D = 29$ such that $De = 2\phi(n) + 1 = 145$

Message $m = 23$

$$Encryp(m) = 23^5 \bmod 91 = 4.$$

$$Decryp(m) = 4^{29} \bmod 91 = 23.\Box$$

Judging solely on the above example, it might not seem obvious that the RSA system is safe. The reason is because the example's numbers are small. As stated earlier, with p and q both being about 130 bits, their product,n, can range in excess of 160 bits. In turn, e and D are also large numbers. Given this kind of range, to crack the code, one must face the discrete logarithm as well as factorization. To date, the factorization of a large product of primes remains unsolvable within a feasible time [Ref 2]. This fact is further examined in the next section, cryptoanalysis.

G. CRYPTOANALYSIS

The art of breaking cryptographic code is called cryptoanalysis. Since there are many public–key systems, the cryptoanalysis of only the RSA system is discussed so as to provide a flavor of how difficult it is and thereby prove its soundness.

The gist behind breaking the RSA system is the ability to solve for both the discrete logarithm and factorization problems. The latter of the two is the most difficult so the discrete logarithm problem will be the first to be explored.

Given the public key $< e, n >$ and let's assume we were somehow able to factor n and therefore know p and q. We can now use Euclid's algorithm the same way as if the sender would to make his/her private key. Take the example in the RSA section.

$$< e, n > = < 5, 91 >$$

Knowing p and q we can compute $\phi(n) = (p-1)(q-1)$

Use Euclid's algorithm to find the secret key D such that

$$De = 1 \bmod \phi(n)$$

With D, the sender's encryption can be intercepted and decrypted by

$$encryp(m)^D \bmod n$$

We have done the easy part. So far we assumed to know the two prime factors of the modulo n in the public key $< e, n >$. The main insurance of the RSA system is the derivation of the two factors p and q [Ref 15]. Whereas the cryptographer only has to come up with two primes, a difficult task but not impossible with the primes being about 130 bits, the cryptoanalyst, in order to recover the two primes to compute $\phi(n)$, must face the grim task of factoring a number in excess of 260 digits within a finite time limit. This leads to the topic of factorization which will also be exploited as the safety basis for the later proposed cryptosytem based on neural network.

1. Factorization

A factorization problem has no current classification but the consensus is that it is neither a Polynomial (P) nor Nondeterministic Polynomial (NP)– Complete problem [Ref 16]. It is loosely described as a Nondeterministic Polynomial Indistinguishable (NPI) problem [Ref 16]. An algorithm is said to run in polynomial time (P) if there are constants A and c such that the running time for all inputs of length

k is Ak^c for all k. All P problems are deterministic and P–time bounded. An algorithm is deterministic if at each step of the computation, the next step is *unique*. P–time bounded means that the execution is in polynomial time since its complexity is bounded by a polynomial in the input length. An algorithm is said to run in NP time if there are no known deterministic P–time solution. In NP problems, at each step of computation, decision problems on the next step exist. To systematically solve an NP problem requires exponential time. A subset of NP problems, an NP–complete problem surfaces when P=NP. NP–complete problems are considered as the most difficult class in NP. An NPI problem is basically defined as having the level of difficulty in between NP and NP-complete. Factorization, an NPI problem, can not be solved in P–time and is not a member of NP–complete [Ref 2].

In order to be convinced that factorization of large numbers is at this time insurmountable, we examine the most straightforward and therefore easiest method. Given a number n to be factorized, we compute $\sqrt{n}$ and round it to the next integer value, m. We then use m as the final index of a for to loop beginning with 1. In each iteration of the loop, the operation (n mod index) is performed until the result is 0 notifying that an integer factor is found. Considering the speed of the computer, this is not a bad method of factorization if n is within a certain range of digits in length. However, this limit is what is exploited in public–key system (n is more than 130 digits in length.) The shortcoming of this method is explored using Matlab program on an IBM '486, 50 MHz, 16 MBytes (Appendix A). The result is shown in Table 2.4.

Undisputably, with n being at least 100 decimal bits in the RSA system, the method above, although possible, is hardly feasible if exhaustive search is required.

Fortunately, the mathematics of factoring have long surpassed the simplicity of the forementioned method. Currently there are established algorithms as well as

Digits factorized	Aprroximate time
10	less than $1msec$
15	$1.5sec$
20	$15min$
25	$28hr$
30	$3yr$ *
40	3000 centuries *

* Estimate

TABLE 2.4: EXHAUSTIVE FACTORIZATION WITH ONE '486 COMPUTER

on–going researches which could reduce the time factor at a phenomenal rate.

As a result of a concerted effort initiated in 1982, the mathematics department at Sandia National Laboratory established some tangible bounds on the computational feasibility of factoring large numbers. The outcome, using a Cray X–MP computer, was within a range of 7.2 minutes to 32 hours for numbers varying from 55 to 77 digits in length [Ref 17].

In a separate study by Ronald Rivest [Ref 15], it is proven that with the best algorithm available such as that of a quadratic sieve [Ref 18], a large prime composite integer can be factored with a running time proportional to:

$$e^{\sqrt{ln(n)ln(ln(n))}}$$

In the range of interest(approximately 256 bits in length), for k bit number n, a crude approximation is:

$$5 \times 10^{9+(k/50)}$$

Using Sandia's benchmark that a 75-digit number can be factored in about 1 day [Ref 17] and the formula of Rivest's article [Ref 15], Table 2.5 is derived [Ref 17].

Based on the data above, it is safe to surmise that the problem of factorization of large number will remain insurmountable for a long time given current

Number of digits	Number of operations	Solution time
75	9×10^{12}	1 day
100	2×10^{15}	255 days
125	3×10^{17}	103 years
150	3×10^{19}	9755 years
175	2×10^{21}	70 thousand years
200	1×10^{23}	36 million years

TABLE 2.5: FACTORIZATION TIME WITH SANDIA'S BENCHMARK [REF 17]

knowledge and technology. The exploitation of this problem in the RSA system and the neural network–based system of Chapter IV is hereby justified.

III. HARDWARE DEVELOPMENT OF THE PUBLIC–KEY CRYPTOSYSTEM

The feasibility of most popular public–key systems is heavily dependent upon the possibility of hardware implementation. Although the algorithm is theoretically simple, its software implementation is slow and highly limited to the resolution of the processor. Such problems are not worth tackling when, with the available VLSI technology, hardware implementation is faster and more efficient.

The crux of many public–key cryptosystems hardware rests on the ability to devise a fast exponentiation scheme where the exponent and modulus are extreme in length (greater than 256 bits). From our two sample cryptosystems, Diffie-Hellman and RSA, the fast exponentiation problem is essential in putting the theory to practice. To familiarize the reader with the possibility for hardware implementation of existing public–key cryptosystems, this chapter will develop in detail a hardware scheme for fast exponentiation based the recursive sum of residues algorithm.

A. MODULO EXPONENTIATION USING RECURSIVE SUM OF RESIDUES

Currently the most popular working hardware for the RSA system performs exponentiation by repeated squaring operations coupled with conditional multiplication. During each square or multiplication stage, modulo reduction is also incorporated so as to maintain a small intermediate result [Ref 19]. The combination of squaring (considered as part of multiplication), multiplication and modulo reduction operations forms the core of fast exponentiation. Currently, there are two categories separating the various methods of implementations:

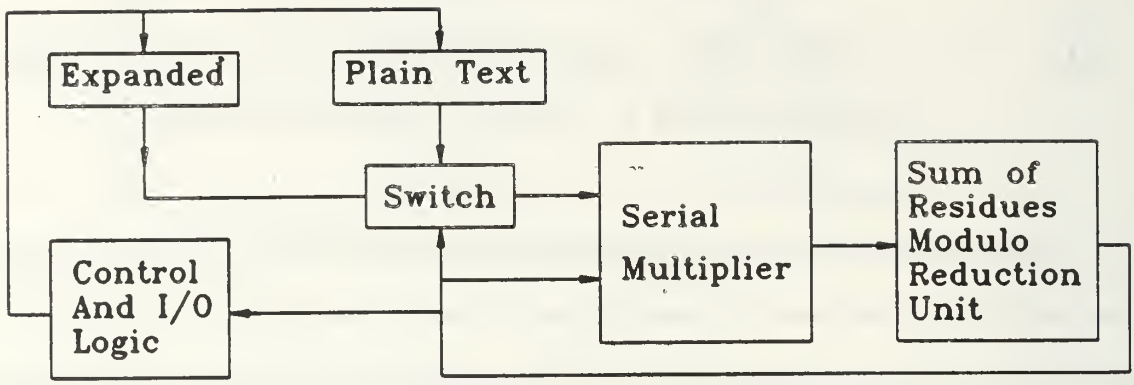

Figure 3.1: Block Diagram of over all exponentiation unit

1. Multiplication and modulo reduction are done in tandem. As the partial products are formed, a decision based on special algorithms is made on whether to perform a reduction on the product [Ref 19].

2. Multiplication and modulo reduction are done sequentially. The result of the multiplication is first obtained and then fed serially to the modulo reduction unit [Ref 19].

For the purpose of this thesis, only the latter case (2) is considered. The underlying reason behind this choice is simplicity which leads to a modular structure that in turn can easily be implemented in VLSI. Moreover, the first part of this hardware scheme, a serial multiplier, will not be delved into with details due to the abundance of such units already available. This leads us to focus on the hardware implementation of the modulo reduction unit to which the result of the serial multiplier is fed into in accordance to the basic block diagram of Figure 3.1 [Ref 19].

1. Sum of Residues Reduction

Our modulo reduction unit is based on the sum–of–residues reduction method. That is the number, x, reduced by modulus, m, is expressed in the following binary form:

$$x = \sum_{i=1}^{n} x_i 2^{i-1};\ x_i = [0,1]$$

The modulo reduction is

$$x \bmod m = (\sum_{i=1}^{n} x_i 2^{i-1}) \bmod m$$

Since modulo reduction is associative

$$x \bmod m = (\sum_{i=1}^{n} x_i (2^{i-1} \bmod m)) \bmod m$$

Summarizing, one performs the reduction as a conditional power of 2 reduced by mod m (a residue) and a summation of all the resulting residues (hence sum of residues) [Ref 19].

Example:

modulus m is 7, $x = 10010 \equiv 18$, i initialized to 1.

Residues are at 2^1 and 2^4 due to positions of 1 in 10010. Respectively the residues are 2 mod 7 and 16 mod 7 which are 2 and 2. Hence $\sum r_i = r_1 + r_4 = 2 + 2 = 4$.

Table 3.1 summarizes the SOR process for the example which resulted in:

$$(\sum r_i) \bmod 7 = 4 \bmod 7 = 4$$

Indeed 18 mod 7 = 4

Given a modulus, residues can be obtained by a look–up table; however, this requires excessive space. Given n as the modulus length, a typical table size is n

shift x LSB First	$\times$	residue 2^{i-1} mod 7	= resulting residue
0	$\times$	2^0 mod 7 = 1	= 0
1	$\times$	2^1 mod 7 = 2	= 2
0	$\times$	2^2 mod 7 = 4	= 0
0	$\times$	2^3 mod 7 = 1	= 0
1	$\times$	2^4 mod 7 = 2	= 2
.	.	. residues will repeat	$\sum$ resulting
.	.	124124...	residues = 4
.	.	pattern	

TABLE 3.1: EXAMPLE SUM OF RESIDUES FOR 18 mod 7

iteration	$2r_{i-1} - m$	$ri = 2r_{i-1}$ or $2r_{i-1} - m$
1		r_1 initialized to 1
2	$2 \times 1 - 7 < 0$	$2 \times 1 = 2$
3	$2 \times 2 - 7 < 0$	$2 \times 2 = 4$
4	$2 \times 4 - 7 > 0$	$2 \times 4 - 7 = 1$
5	$2 \times 1 - 7 < 0$	$2 \times 1 = 2$
.	.	.
.	.	.
.	.	.

TABLE 3.2: EXAMPLE RECURSIVE SOR FOR 18 mod 7

by $2n$. With n being greater than 256 bits, this would require extremely large data paths, undesirable in silicon implementation [Ref 19]. For this reason, it would be more desirable to calculate the residues as necessary in accordance with the given modulus. Fortunately, there is a simple recursive formula which allows for easy hardware calculation of residues:

ith residues $\equiv r_i;\ i = 2...n$

$$r_i = \begin{cases} 2r_{i-1} & iff \quad (2r_{i-1} - m < 0) \\ 2r_{i-1} - m & iff \quad (2r_{i-1} - m \geq 0) \end{cases}$$

r_1 initialized to 1 [Ref 19]

Taking the previous example from Table 3.1 and incorporating into it the

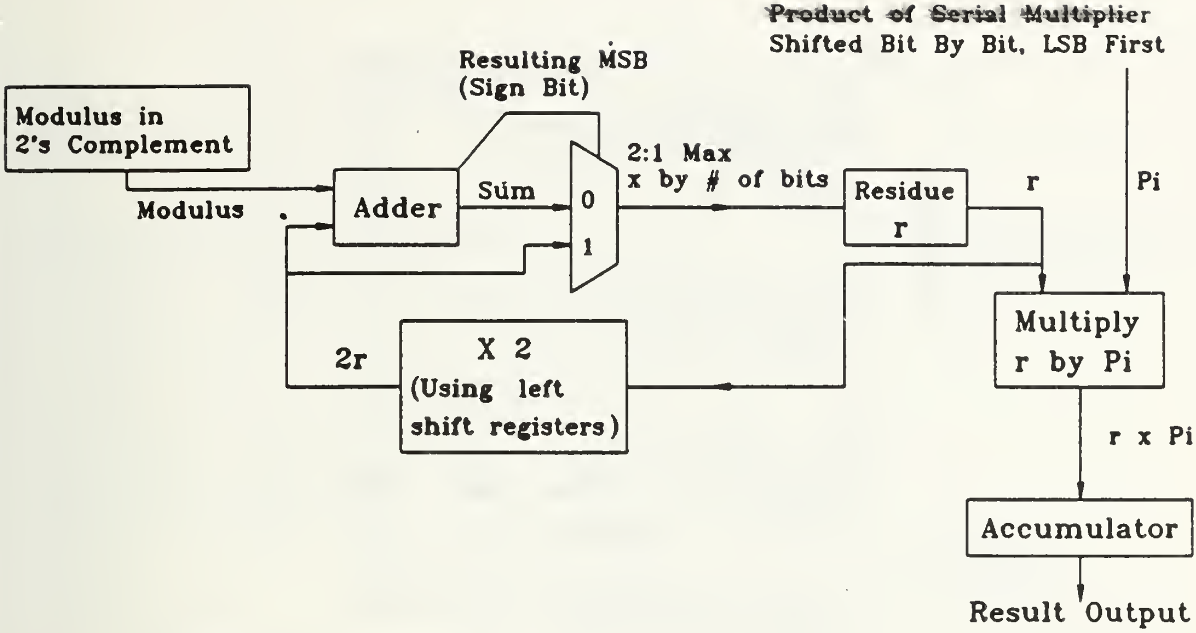

Figure 3.2: Modulo Reduction Unit

recursive sum of residues method, the result of which is in Table 3.2, indeed the residues are the iterative pattern: 1,2,4,1,2,4,1...

A diagram of an architecture using the sum of residues method for modulo reduction is provided in Figure 3.2 [Ref 19] .

Respectively, M and R are two n-bit registers holding $(-m)$, the two's complement of the modulus, and r_i, the current residue. Initially, the current residue is set to 1. As the system is clocked, the register is loaded with $2r_i$ or $2r_i - m$, depending on the sign bit of the $2r_i - m$ add. The accumulator sums those residues which are passed by the incoming bits of the serial multiplier's product P. There's an overhead amount of bits which must be taken into acount for the accumulator's size. The necessary overhead bits are given in Figure 3.3 [Ref 19].

Having a sound understanding of the theory behind the architecture in Figure 3.2, the next obstacle that must be cleared is the transformation of the theory to an actual VLSI layout. With some intuition and basic knowledge of logic circuit, a block diagram complete with logic units, inputs and outputs is developed and shown

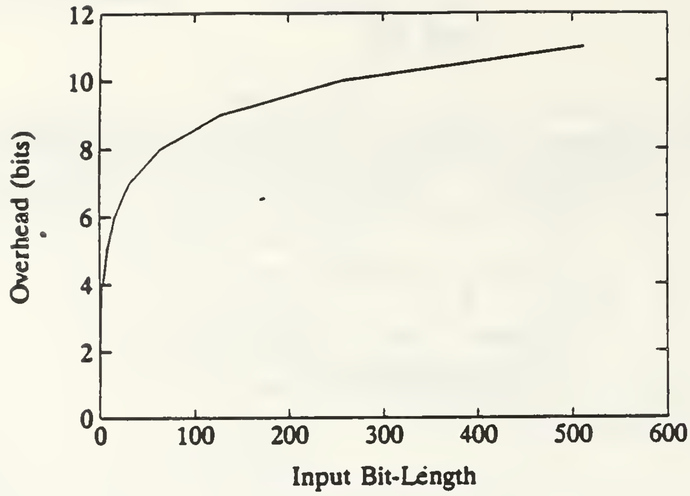

Figure 3.3: Overhead Vs Input Bit

in Figure 3.4.

A few details in the transformation between Figures 3.2 and 3.4 are hereby stated for clarification. Whereas in Figure 3.2 a multiplier was used to obtain the correct residue for the accumulator, in the final design, a multiplexer is chosen to perform the multiplication. Also the left shift logical to obtain $2r_i$ is finalized without a shift register but rather by hardwiring the outputs of the residues directly to the inputs of the first adder.

From a VLSI perspective of Figure 3.4, one sees that it is beneficial to devise a modular unit (shaded region) which could easily be assembled together to form a larger complete reduction unit satisfying the length of the modulus. To realize a single modular unit, only 2 master–slave flip flop's (MSFF), 2 combinational adders and 2 2:1 multiplexers are needed. The control for this unit alone and for the rest of the modular reduction device is a couple of simple two-phase clocks. The simplicity of this modular scheme is attractive. However, the cost is in silicon area and speed as we will see.

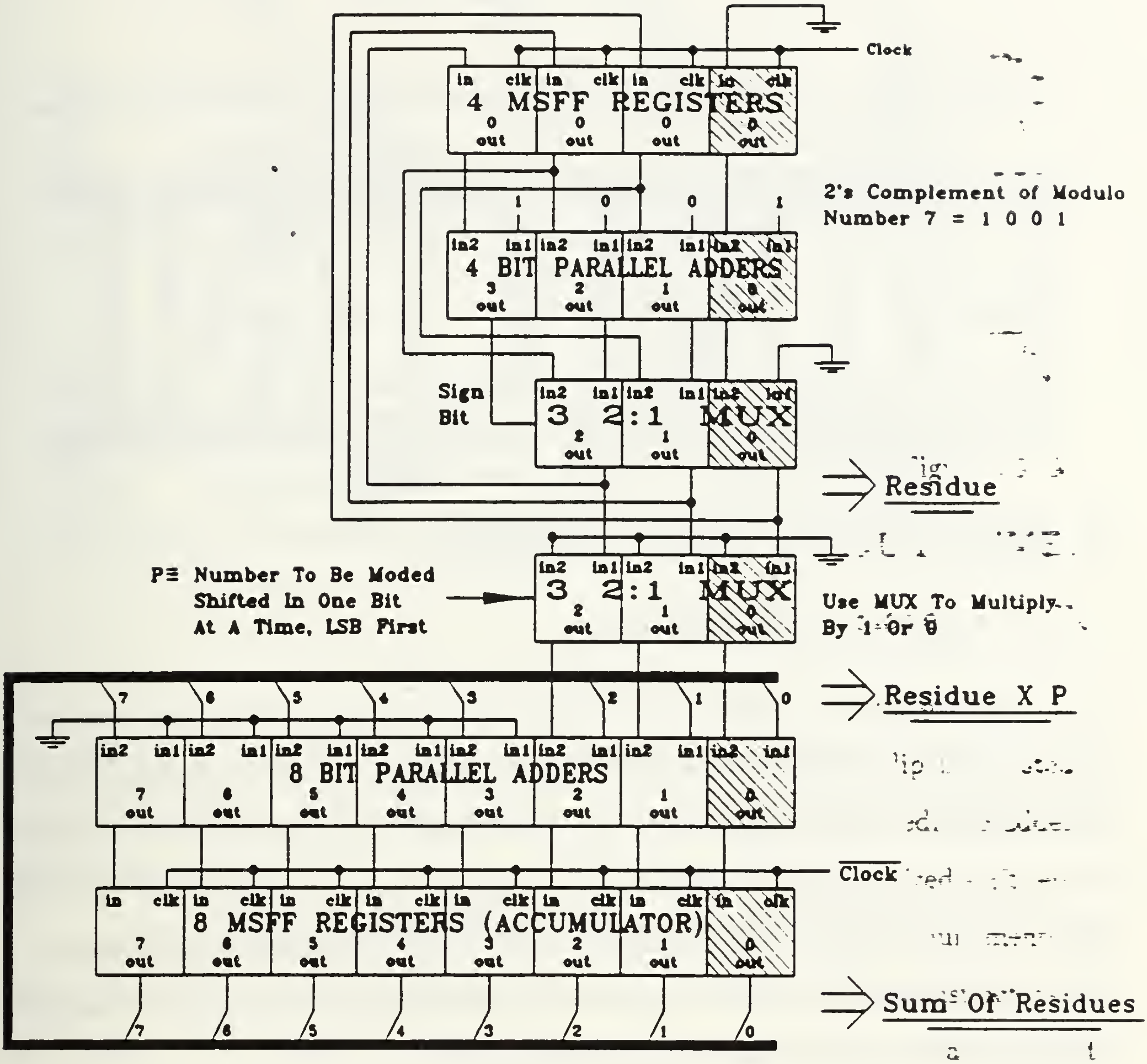

Figure 3.4: Block Diagram of 4-Bit SOR with Logic Units

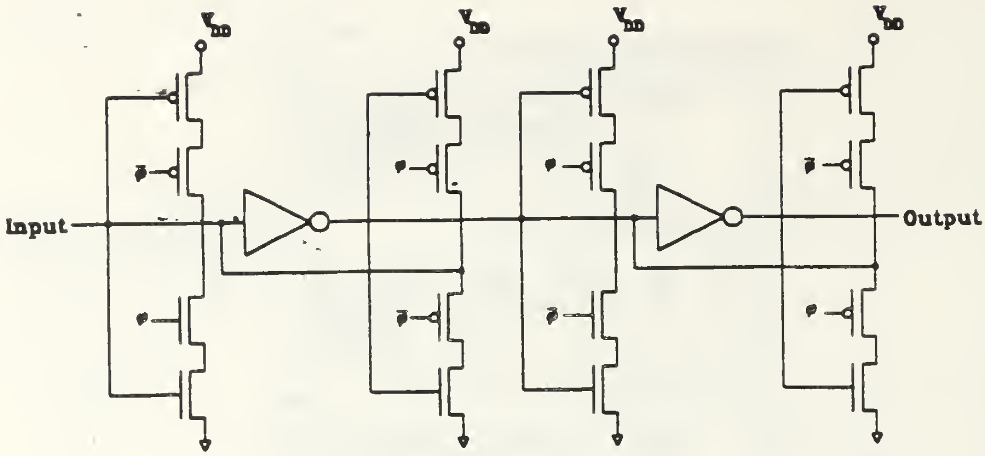

Figure 3.5: MSFF Circuit Diagram

B. VLSI LAYOUT DEVELOPMENT

1. Master Slave Flip Flop

The desire for a simple control method, a two–phase clock, necessitates the use of a master–slave flip flop instead of a direct latch. In the first stage where the residues are computed, the adder uses the output of the flip flop (slave) while the output of the hardwired shift left $2r_i$ is transferred to the input end of the flip flop(master). The same requirements for the flip flop are imposed in the accumulator unit where the flip flop must act as both the accumulator's adder output register (master) as well as accumulated input to the adder.

The chosen circuit for our master–slave flip flop is shown in Figure 3.5 [Ref 20].

Analysis of Figure 3.5 shows two cascading 2–phase static latch. This structure is sound and efficient to implement. A minor problem of clock race is possible when clock is high and clockbar overlaps it causing a tendency for the input and feedback signal to contest with the new value on the flip flop input [Ref 20]. Fortunately, for our purpose, this problem did not manifest itself as the feedback transistor is designed to "trickle": transistor β is low [Ref 20]. The VLSI layout for the master–

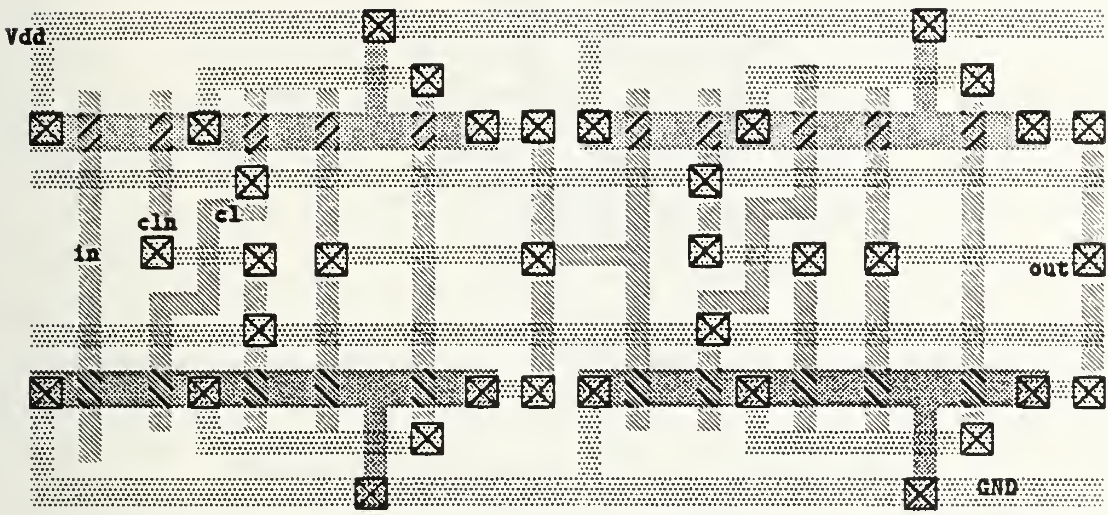

Figure 3.6: MSFF Layout

slave flip flop is given in Figure 3.6. It should be preempted that the design will be slightly alter later on in order to conform to the overall modularity of the entire modulo reduction unit.

Silicon space for the MSFF is $64 \times 135\ \mu m^2$. SPICE analysis [Ref 21] on the layout determined a delay from input to output to be $10ns$. The maximum speed of operation for the MSFF is 100Mhz. Since the input and output of the MSFF is inherent only to the single module, no effect from the other modules are of concern.

2. Adder

Due to the modularity of the design, the simplest approach is taken in the development of the two adders in the module. The chosen unit for both adders is a combinational adder with approximately equal sum and carry delays. Carries are allowed to ripple through the necessary modules. This choice is made mainly to conform to the modular structure. The ripple carry design does cost much in speed. The circuit diagram for the adder is shown in Figure 3.7 [Ref 20]. The appropriate

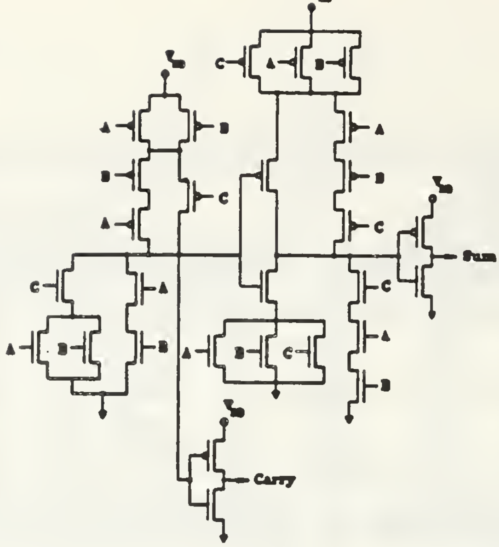

Figure 3.7: Adder Circuit

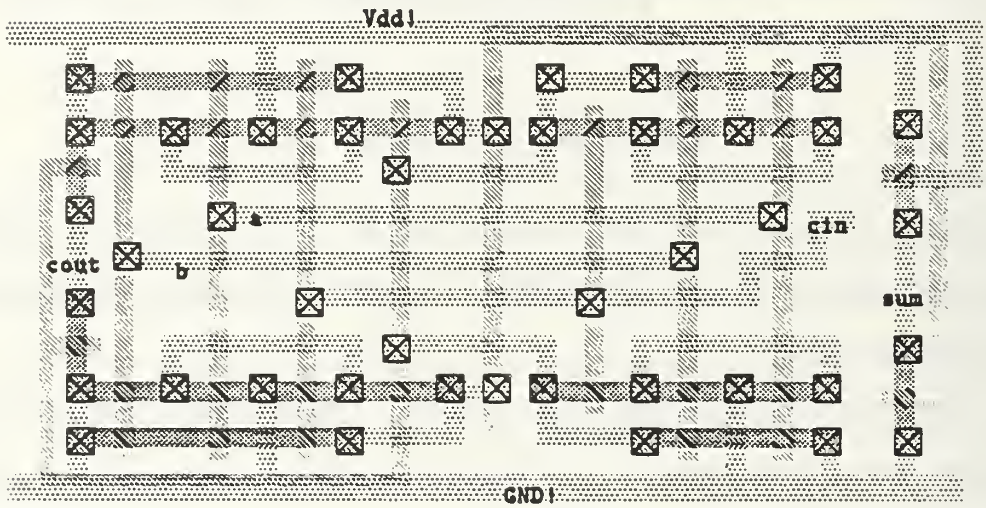

Figure 3.8: Adder Layout

layout follows in Figure 3.8.

The adder layout sizes up to $73 \times 145\ \mu m^2$. SPICE analysis Ref 21] of a single adder unit showed that the sum and carry delays are $4.8ns$ and $4.5ns$ respectively. From this result, intuition dictates that when the unit is put together for a larger modulus, the carrychain will be the limiting parameter for speed of operation.

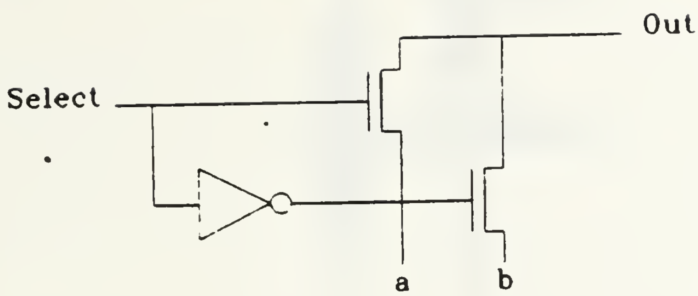

Figure 3.9: MUX Function Block Circuit Diagram

3. Multiplexer

The reduction unit calls for the use of two 2:1 mux's per bit of modulus. The first takes its select input from the sign bit of the sum of the first adder and output $2r_i$ or $2r_i - m$ as appropriate. The second simply acts as a multiplier with its select input as the single bit shifted in from the output of the serial multiplier and outputs the residues if the select is 1 and 0 if select is 0. In short it acts as a single bit multiplier. For our multiplexer, a function block design is used [Ref 22]. The circuit is shown in Figure 3.9 [Ref 22].

This is an NMOS device in which only one of the two inputs a, b is passed to the output depending on whether NMOS-1 or NMOS-2 is turned on. Only one NMOS gate can turn on at the time because the inputs to their gates are complements. Intuitively, the select input of the multiplexer is the input to the two gates. The VLSI layout is shown in Figure 3.10.

Because of the simplicity of the circuit, the only delay is one transistor gate. Compared to the delay of the adder or flip flop, this is negligible and will not be delved into. The size of the layout is $32 \times 33\ \mu m^2$.

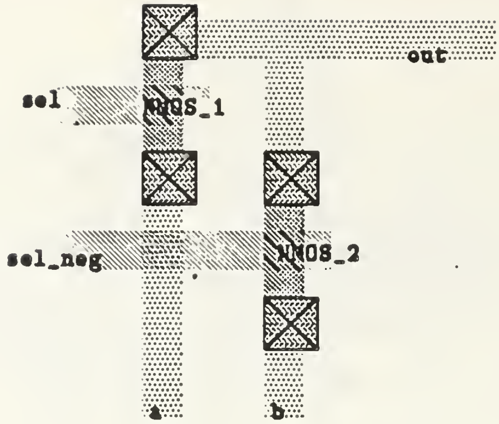

Figure 3.10: Layout of MUX

4. Modulo Reduction Unit

Having all the necessary components, the entire modulo reduction unit can now be developed. As previously mentioned, a "modular" design is implemented in this thesis so that, depending on the size of the modulus, the entire unit can be constructed by simply cascading the same module together n times (modulus is n-bit in length.) Bearing this in mind, the layout for the module as well as a 4-bit modulus modulo reduction unit is shown in Figure 3.11.

The foremost significance of the VLSI scheme for the modulo reduction unit is that it is simple in implementation and, above all, it works. Using a CFL program [Ref 3], the module can easily be generated into an n bit unit. Experimentally, RNL simulations were performed [Ref 3]. The results, which are enclosed in Appendix B, testify strongly on behalf of the unit's functional capability. However, as to the efficiency in area and speed, the empirical data is debatable in support of different individual's needs.

Since the modulo reduction unit is designed mainly for modularity, the size of the entire structure grows geometrically with the number of bit that the unit is designed for. Each module per bit is sized at $73 \times 672\ \mu m^2$. If n is the number of bits required to be modulo reduced, then n modules are needed. Disregarding the

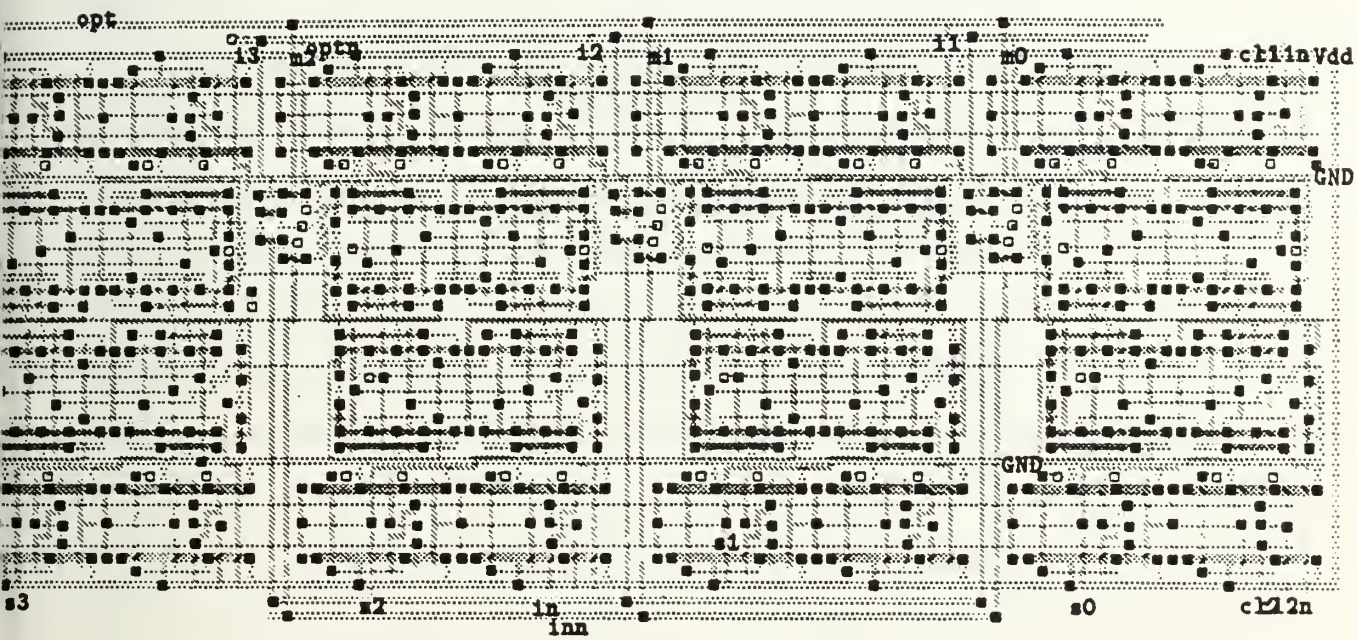

Figure 3.11: Layout of 4-bit Modulo Reduction Unit

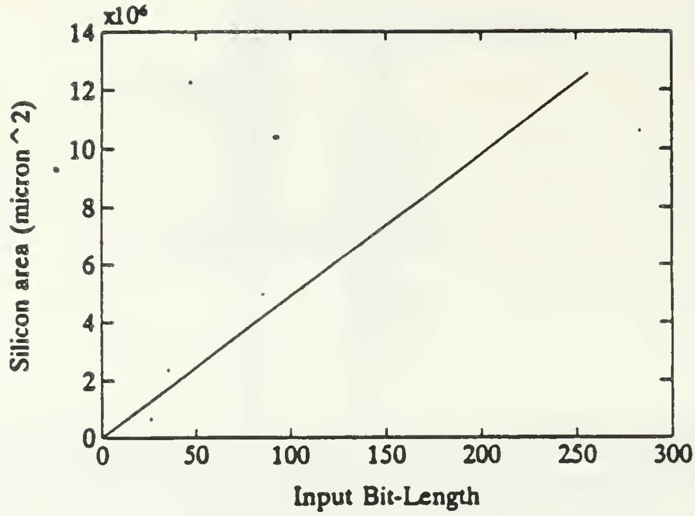

Figure 3.12: Size of Modulo Reduction Unit

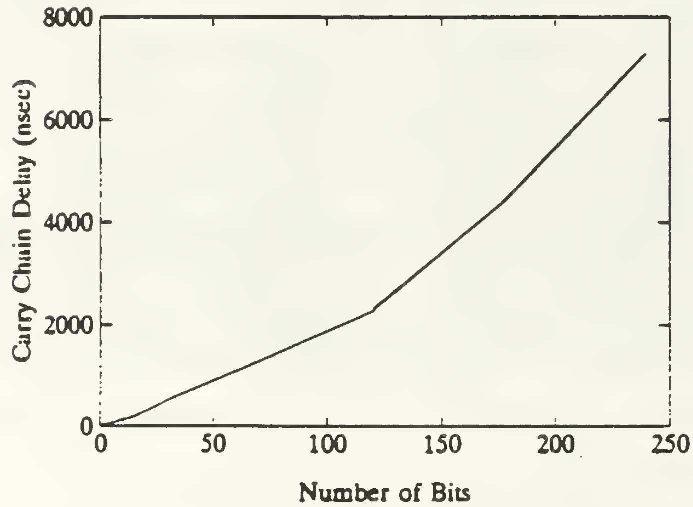

Figure 3.13: Speed Performance of Modulo Reduction Unit From SPICE

minimal effect of overhead bits (Figure 3.3), the size of a modulo reduction unit for n-bit modulus is $n \times 49056\mu m^2$. Figure 3.12 is a plot relating the size of the unit to the number of bits.

In regard to speed consideration, experimental data found the unit's carrychain to be the limiting factor. After SPICE simulation [Ref 21], Figure 3.13 was obtained to gauge the speed performance of the modulo reduction unit.

Since the carrychain imposes the speed limit in this design, intuitively, one can incorporate speed saving techniques such as various carry–look–ahead adders; however, this will alter the modularity structure. This is beyond the scope of the thesis but remains a viable avenue for speed improvement at the expense of silicon space.

In summary, this chapter has provided the basic hardware building blocks for a fast exponentiation scheme with specific details on a modulo reduction unit. From this foundation, an RSA hardware implementation can easily be conceived. Such an implementation is necessary in many applications, one of which is the subject of the next chapter: a novel approach to PKS using neural networks. As will be explained in the following chapter, the hardware technology developed here will be a small integral part of a "pseudo" public–key cryptosystem based on neural networks.

IV. A NEURAL NETWORK–BASED PUBLIC–KEY CRYPTOSYSTEM

Since all cryptosystems make use of some form of mapping functions to transform data to unintelligible code and then recover it, a neural network – inherently an excellent non–linear mapping technique – provides a viable choice for a medium from which a possible cryptosystem can be based upon. In examining this possibility, this chapter presents an adaptation of the back–propagation neural network to a "pseudo" public–key arrangement. Strictly as an initial research, a simple requirement of encrypting and decrypting a number representing any character or data is fulfilled via the network. Following examinations of the network, a key management system is then devised. As data are fed to the network in simulation of encrypting and decrypting, the problems and solutions to the system are discussed. Finally, a complete top–down block diagram of an entire cryptosystem based on the neural network of this study is proposed.

A. EXPERIMENTS IMPLEMENTING A NEURAL NETWORK IN CRYPTOSYSTEMS

The neural network–based cryptosystem to be designed, a cipher system, requires two basic elements: a key management scheme and an algorithm for two–way mapping a set of numbers representing data. In this respect, it is fundamentally not far different than other cryptosystems. The differences surface only in the implementation of mapping. Whereas all existing system such as DES [Ref 23], once implemented in hardware, maps in a *set* pattern, a neural network can change its mapping any time by simply retraining its weights to new data. As it turns out, this deviation from the norm is advantageous since it adds an extra level of protection. Namely, if the system is compromised, retraining and obtainment of new weights are neither a difficult nor time–consuming task [Ref 24, 25].

Before the network is presented, some background is in order. The system of this study is designed to map up to a set of 45 characters for encryption and decryption. Figure 4.1 is a block diagram of the system. From Figure 4.2 [Ref 26], the two networks for encryption and decryption are identical systems; they are both back–propagation networks composed of 4 inputs, 1 output, and three hidden layers of various sizes.

Prior to proceeding with the explanations of Figure 4.1, it is stressed that this system is based mainly on the RSA system. As such, it simply takes a *number*, encrypts it to another *number* and decrypts it back. Like RSA, this is all the neural network is set up to do. For simplicity, this number represents a particular character; however, the relationship between the number and character is not explored in detail because this is a subject outside of the focus of this thesis. Furthermore, the input to the network of this research is only 16 bit in length. Again this is chosen for simplicity and clarity in an example system. It is not chosen for security. Like RSA in which system security rests on the key being numbers greater than 256 bit, the security of this system also depends upon the range of the input being greater than 256 bit. In fact, with the input being only 16 bit long, the system can be compromised within nanoseconds. However, successful cryptoanalysis of 256-bit inputs will be shown in Section 4.D.1 to take trillion of milleniums. So in order to apply this system to real–world application, it is preempted that the input range should be increased and the assignment of a number to character be done separately so as to maximize security.

To clarify Figures 4.1 and 4.2, in order to encrypt, a 16-bit number representing a character is partitioned into 4 segments so as to provide the 4 4-bit inputs to the

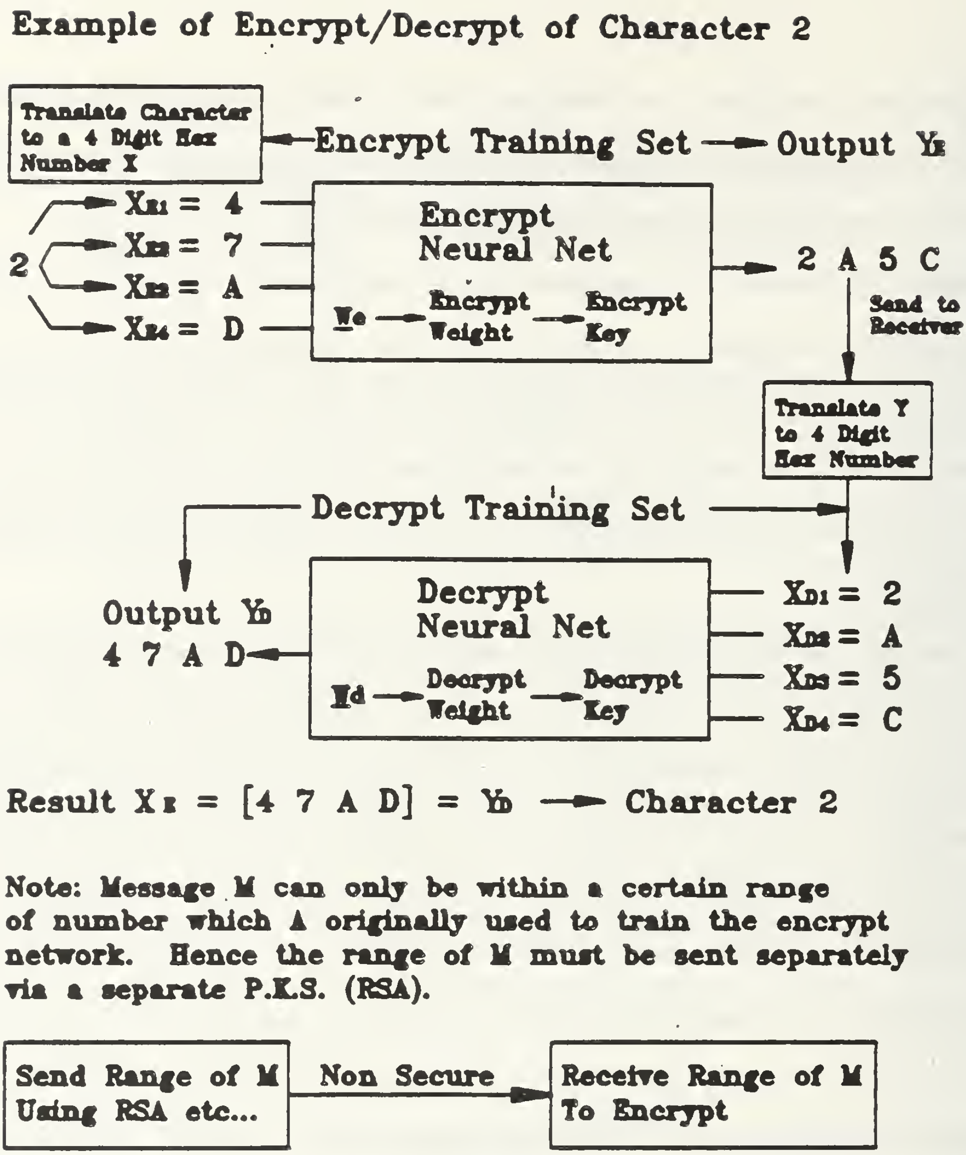

Figure 4.1: Neural Network As A Cryptosystem Block Diagram

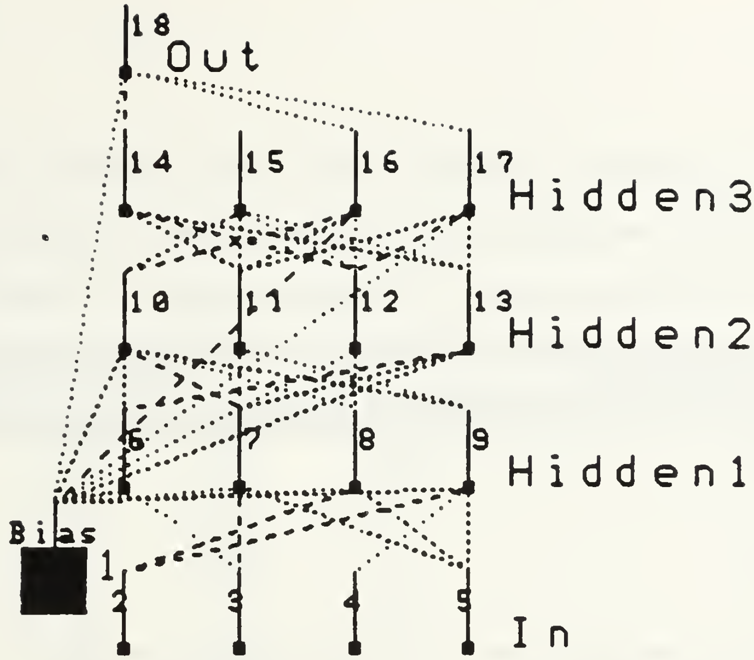

Figure 4.2: Back–Propagation Network For Encryption and Decryption

encryption network, the output of which is a single 16-bit number different than that of the original input. These 4 4-bit inputs along with their corresponding 16-bit output are first fed to the network to train the weights. Once trained, the weights of the encryption unit would have converged to values such that when these converged weights are set as constants, the same 4 4-bit inputs used for training will provide an actual output that can be *rounded* to the desired output used in training. For example, if the desired output is 1256 then the actual output must be between 1255.5 and 1256.5 so that rounding to the nearest integer would yield 1256.

Naturally, for a system encrypting up to 45 separate characters, the corresponding training sets will be 45 input/ouput pairs. Basically, this is how the network is trained and utilized for encryption. It should be noted that whether the input/output pairs are linearly related or not, the weights should converge and accommodate the required mapping function.

For decryption, the same type of network, training and mapping scheme will be used, only this time the recovery of the original data is essential. Intuitively, the input of the decryption unit is the 16-bit output of the encryption network. To keep the structures of the encryption and decryption networks identical, the encryption output must be partitioned into 4 4-bit segments before it becomes inputs to be decrypted. The desired output of the decryption network must then be the original 16 bit input of the encryption network. To clarify the process, the following example is offered.

Example A:

Given a single processing element with 4 inputs and one output.

The element's function is $f(\sum) = \sum$;

The four input x's= $[1\ 2\ A\ 6]_{16}$; output=12599 $\equiv 3137_{16}$

The four converged encryption weights are found to be [**77 1056 501 900**] such that

$$1(\mathbf{77}) + 2(\mathbf{1056}) + 10(\mathbf{501}) + 6(\mathbf{900}) = 12599.$$

The encryption weights are thus : [**77 1056 501 900**].

Since the encrypted output is 3137_{16}, the decryption input is $[3\ 1\ 3\ 7]_{16}$

The four converged decryption weights are found to be [**290 66 997 121**] such that

$$3(\mathbf{290}) + 1(\mathbf{66}) + 3(\mathbf{997}) + 7(\mathbf{121}) = 4774 = 12A6_{16}.$$

The decryption weights are thus : 290 66 997 121. □

Based on the example, a training set of several encryption and corresponding decryption numbers can be randomly picked to represent any character. A typical training set for 28 characters, the upper case alphabet with comma and space, is shown in Table 4.1.

Encryption $\Longrightarrow$			$\Longleftarrow$ Decryption		
Text Character	Hex Rep	Dec Rep $\rightarrow$	$\leftarrow$ Encrypted Character	Hex Rep	Dec Rep
A	12AC	04780	R	321C	12828
B	134E	04942	N	981B	38939
C	214B	08523	P	A235	41525
D	2698	09880	S	425A	16986
E	35B7	13751	Q	6533	25907
F	538A	21386	O	A159	41305
G	6942	26946	L	8731	34609
H	661B	26139	D	2698	09880
I	728D	29325	M	9137	37175
J	7546	30022	H	661B	26139
K	811A	33050	B	134E	04942
L	8731	34609	J	7546	30022
M	9137	37175	C	214B	08523
N	981B	38939	F	538A	21386
O	A159	41305	A	12AC	04780
P	A235	41525	G	6942	26946
Q	6533	25907	K	811A	33050
R	321C	12828	I	728D	29325
S	425A	16986	E	35B7	13751
T	B366	45926	Z	F553	62803
U	B129	45353	Y	EA54	59988
V	C568	50536	space	0BCA	03018
W	D346	54086	U	B129	45353
X	D351	54097	W	D346	54086
Y	EA54	59988	V	C568	50536
Z	F553	62803	comma	092D	02445
space	0BCA	03018	X	D351	54097
comma	098D	02445	T	B366	45926

TABLE 4.1: EXAMPLE TRAINING SET

Notably, the assignment scheme of Table 4.1 is *monoalphabetic*. This is chosen strictly for simplicity, not security. The focus of of the neural network is to map a number to another then recover it. How the number might represent a character is entirely another subject in cryptography. In light of this, using training sets similar to Table 4.1, experiments were next conducted to support the proposed theory of using neural networks for a cryptosystem.

B. EXPERIMENTAL RESULTS AND OBSERVATIONS

In order to accommodate the mapping scheme for the proposed cryptosystem, a series of experiments designed to gauge the performance of the back–propagation network were carried out. The primary goal of the experiments is the development of an optimal network based on several parameters. Information such as training time, error tolerance, range of input numbers, network sizes and their interdependence are of primary interest in building a working example network for the cryptosystem. In accomplishing the desired goal, the chosen back–propagation network consists of 4 inputs, 1 output and 3 hidden layers of various sizes. The network is built and simulated using the Neuralware software package [Ref 26] implemented in an IBM '486, 50MHz, 16 Mbytes.

Table 4.2 provides the first set of results which are intended to show the relationship between convergence error and training time. For the experiment, a set of 45 training input/output pairs (45 characters of NTP) along with 4 bit per input (16 bit overall since there are 4 inputs) were used. Error is measured in root mean squared values (RMS), a common statistical method of error estimation which is employed by Neuralware. Training time is compared by number of iterations, a method of measurement used in Neuralware. It should be noted that time of iterations varies for different networks. The larger the network, the time per iteration

Number of Elements per Hidden Layer	Iterations →	RMS Error	Iterations →	RMS Error
5	2500	0.6	250000	0.5
10	73000	0.0025	300000	0.002
15	70000	0.002	350000	0.00006
20	124500	0.0005	270500	0.0001
25	115570	0.000085	340000	0.000017

TABLE 4.2: TRAINING TIME VS ERROR RELATIONSHIP

increases proportionally.

Conclusions drawn from Table 4.2 concern primarily training time and error. Comparing the error with iterations to the error, one noted that up to the first set of iterations, the errors decreased significantly for all networks. After this, the error goes down significantly less even for a greater increase in iterations. This shows that after a certain barrier, training of all networks follows the law of diminishing return wherein the error decreases minimally despite greater increase in training time. Eventually, when the error has reached its minimum, no amount of training time will help. This behavior is typical of all neural networks [Ref 24, 25]. After this first observation, another set of experiments were run and their results are summarized in Table 4.3. For this experiment, the iterations to convergence were set to 3.5×10^5 iterations where it was determined that the error was at its minimum for all tested networks (weights have converged to optimal values). The inputs again are 4 bit each and 45 input/output pairs were used as training sets.

Clearly from Table 4.3, given the same set of input/ouput, the larger network results in the least error at final convergence. This is due to the larger amount of processing elements and weights (memory) available to accommodate the necessary mapping patterns.

The final experiment intends to formulate the interdependence between network

Elements/hidden layer	RMS error
5	0.2109
10	7.835×10^{-4}
15	3.0836×10^{-5}
20	2.492×10^{-5}
25	1.684×10^{-5}

TABLE 4.3: RELATIONSHIP BETWEEN NETWORK SIZE AND ERROR

size, iterations to convergence, and input size. The results are depicted in Figure 4.3.

The conclusions which can be drawn from Figure 4.3 are:

- In regards to the range of inputs, as the number of bits per input increases, the training time increases. Theoretically, this trend can be attributed to the weights having to accommodate mappings of larger number to smaller ones as well as the reverse. Namely, as a set of small *and* large inputs maps to larger *and* smaller outputs respectively, the weights have to be small as well as large if there are not enough weights. This may lead to non–convergence as they can not be both. This is seen in the extremely high increase in training time with the smaller size networks. As the network grows, there are more weights to map thus there is less strain on the system causing training time to decrease.

- In regards to the number of input/output pairs to be mapped, as the training pairs increased to 45 (number of characters in NTP set), the iterations to convergence also increased. This is easily explained by an analogy to the human brain which is the structure emulated by neural networks. When there is more information to learn, the brain labors to maximum capacity until its cells are depleted. In the case of neural networks, as the size of the network is exceeded by the information memory demands, the iterations increase with approximately no learning. A barrier is reached until more neurons are available.

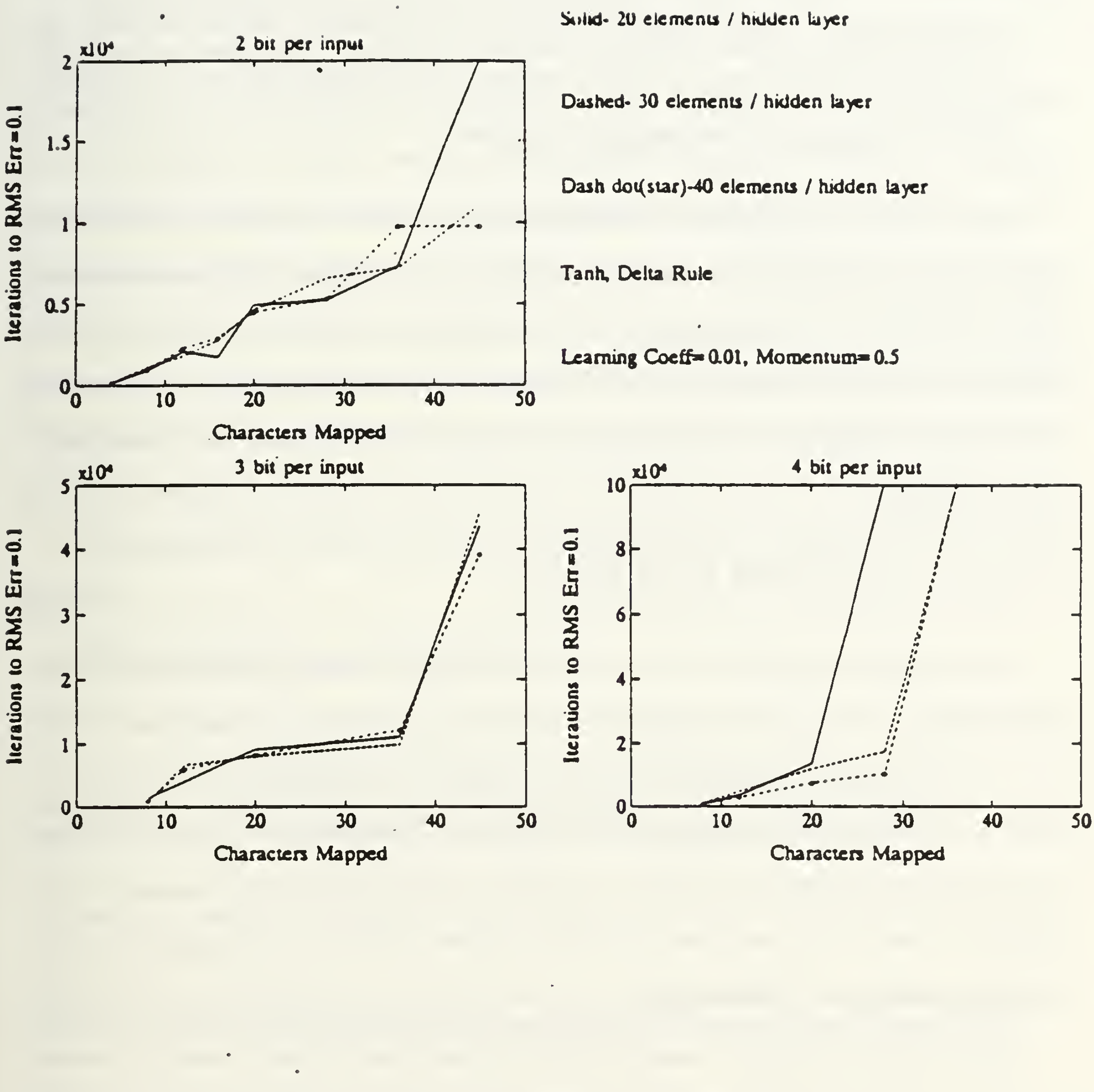

Figure 4.3: Relationship between Network Size, Iterations to Convergence and Input Size

- In regards to the size of the network, the relationship to input/output as well as range of inputs are already described in observations of Table 4.2 and 4.3. One more observation is added here in that as network size is enlarged for more training input or input size, the training time increased. Mathematically this makes sense since there are more weights and neurons (memory) to update. Each iteration now takes longer to complete.

After thorough exploration of empirical data, the final conclusion is that there exists a network for the proposed cryptosystem. And *it works.* After several trials, the optimal network for this paper's system is found to consist of a 4 bit per input, 4 inputs, 1 output, 3 hidden layers, 25 elements per hidden layer, with 45 sets of input/output traing pairs. This specific network is used in a conclusive example in the next section.

C. AN IN-DEPTH EXAMPLE

This example is based on Table 4.1 which in turn is based on the Naval Tactical Publication coding scheme wherein a character is mapped unto another: A↔R, B↔N... This scheme is chosen for clarity in that an encrypted text will also be a string of characters. In reality, however, since the characters are coded by a number, the encrypted text need not be a number representing another character. For instance, character 'A' encrypts to $5BCF_{16}$ where $5BCF_{16}$ in this case does not represent a character in Table 4.1.

This example employs a monoalphabetic substitution scheme to assign a number to a character. In this respect, this system is vulnerable to single–letter frequency analysis and is therefore easy to break [Ref 27]. However, if each character is coded by multiple numbers utilizing schemes such as homophonic or polyalphabetic substitution (Beale or Vignère and Beaufort cipher), the safety margin would greatly increase [Ref 27]. Additionally, for real-world application, the input range must be raised from 16 bit to greater than 256 bit.

As stated in the previous section, this system, based on RSA, is concerned only with two-way mapping a number to another. Bearing this in mind, this section is intended only as a *pedagogical* example of how such a scheme could be implemented so as to be able to actually encrypt and decrypt a plaintext message. In reality, for complete security, a separate scheme of assigning numbers to characters must be chosen to defeat the frequency of letters in plaintext. If interested, the reader is referred to reference 27 for the assignment of numbers to characters. Moreover, the range of the network's input must be greater than 256 bit. Having established the objective of this example, illustrations of the system is hereby offered. The following plaintext message is encrypted and decrypted using the system of Figure 4.1.

Plaintext: FIND ME COMPLETE CHAOS AND I WILL SHOW YOU SCIENCE

Decimal coded text and encrypted text:

```
                   F     I     N     D           M     E           C
                   |     |     |     |           |     |           |
Plaintext:       21386|29325|38939|09880|03018|37175|13751|03018|08523|
Encrypted text:  41305|37175|21386|16986|54097|08523|25907|54097|41525|
                   |     |     |     |     |     |     |     |     |
                   O     M     F     S     X     C     Q     X     P
-----------------------------------------------------------------
  O     M     P     L     E     T     E           C     H     A     O     S
  |     |     |     |     |     |     |           |     |     |     |     |
41305|37175|41525|34609|13751|45926|13751|03018|08523|26139|04780|41305|16986
04780|08523|26946|30022|25907|62803|25907|54097|41525|09880|12828|04780|13751
  |     |     |     |     |     |     |     |     |     |     |     |     |
  A     C     G     J     Q     Z     Q     X     P     D     R     A     E
-----------------------------------------------------------------
        A     N     D           I           W     I     L     L
        |     |     |           |           |     |     |     |
03018|04780|38939|09880|03018|29325|03818|54086|29325|34609|34609|03818|
54097|12828|21386|16986|54097|37175|54097|45353|37175|30022|30022|54097|
  |     |     |     |     |     |     |     |     |     |     |     |
  X     R     F     S     X     M     X     U     M     J     J     X
```

```
-------------------------------------------------------------------------
  S      H      O      W             Y      O      U
  |      |      |      |             |      |      |
16986|26139|41305|54086|03818|59988|41305|45353|03818|
13751|09880|04780|45353|54097|50536|04780|59988|54097|
  |      |      |      |      |      |      |      |      |
  E      D      A      U      X      V      A      Y      X
-------------------------------------------------------------------------
  S      C      I      E      N      C      E
  |      |      |      |      |      |      |
16986|08523|29325|13751|38939|08523|13751
13751|41525|37175|25907|21386|41525|25907
  |      |      |      |      |      |      |
  E      P      M      Q      F      P      Q
```

Resulting encrypted text:

OMFSXCQXPACGJQZQXPDRAEXRFSXMXUMJJXEDAUXVAYXEPMQFPQ

Additionally, given the monoalphabetic scheme chosen here, in order to guard against the problem of frequent repetition in the english vocabulary such as the word *the*, double patterns *ll, nn, tt* which can simplify cryptoanalysis, random or strategically placed noise can be added to the encryption via some algorithm. Remember that since one is using only 28 numbers out of 2^{16} here, there are multitudes of numbers left to insert into the above patterns as noise bytes. In this specific example, the noise is inserted by human intuition and is shown as asterisk (*) signifying any number not used in coding the characters.

An example of encrypted text with noise inserted:

OMFS*XCQX*PACG*JQZ*QXQDR*AEX*RFSXMXU*MJ**JXE*DAUXV*AYXEPM*QFPQ

With the noise option, one must have a scheme to filter the noise out prior to entering the decryption network. The decryption network simply recover the plaintext from the encrypted text as previously discussed. Both the encryption and decryption networks is subjected to the following parameters:

- Momentum coefficient = 0.300.
- Learning coefficient = 0.500.
- Function ≡ Tanh.
- Learning rule ≡ Delta-rule.
- Size ≡ 4 inputs, 1 output, 3 hidden layers, 25 elements/layer.
- The time to minimum acceptable error was approximately 8 hours.

The two networks' (encryption and decryption) data employed for this example are included in Appendix C.

Clearly, the basis of how to encrypt and decrypt via a neural network is established. Based on knowledge of cryptography, the concept of a key must now be incorporated.

D. KEY MANAGEMENT

Up until present, the method of mapping has been discussed without any mentioning of a key. In reality, the key evolves from the actual training process. Namely, once the training is done, both for encryption and decryption, the converged weights are the keys. Since different training sets are used (inverse sets), a key for encryption and another for decryption are required. The keys will change when the network switch mapping function via new training sets.

For our example of only one training input/ouput pair and one processing element in Section A (Example A), the keys are [77 1056 501 900] for encryption and [290 66 997 121] for decryption. The fact that two keys must exist is perhaps clearer now with the example; however, the fact that this is a one-way scheme only remains murky. Let's clarify this further. For a specific set of encryption/decryption key that party A obtains from training, party B given the encryption key, can encrypt while A can decrypt using decryption key. Unless B somehow also obtain the decryption key (the only safe way to do this is through a secured channel) there is no way for A to encrypt to B unless B had come up with separate encrypt/decrypt keys of his own and sent A the encryption key. There is no restriction against both parties using the same encryption/decryption keys that only one has derived, provided the system is a secret–key type where the keys can be distributed through safe channels. In this respect, there is little to gain from a neural network as it is nothing more than another mapping method. But there is much more to the versatility of neural network which should be exploited.

In the key management scheme thus far mentioned, only one party needs to train the network and then passes the weights as keys for encrypt and decrypt to his or her counterpart. However, if both parties were to obtain separate training sets and thus keys, only the encryption keys need to be exchanged. In this respect, there exists a "pseudo" public–key scheme which can be exploited since the decryption key requires no exchange. This possibility is hereby explored.

1. A Proposed Pseudo Public–Key Cryptosystem Using A Neural Network

Irrefutably in cryptography, the possibility of a pseudo–public-key implementation of a neural network merits this paper further examination. Currently, the designed networks mentioned that the keys, the encryption/decryption weights, can be passed through a secured channel. If a cryptoanalyst has the keys and the same network, he has broken all codes. Now the assumption is lifted. This research postulates that if both parties develop their own set of keys, the *encryption* keys can be exchanged through any public channel(Figure 4.1). A cryptoanalyst having possession of the encryption key, a network, and encrypted data will face an enormous obstacle in breaking the code: time (in terms of centuries.)

From the forementioned implementation, one recalls that only the encryption key needs to be exchanged if both parties train on separate data and each obtains his or her own keys. The decryption key is never divulged. Given the encryption key E_{encr} and the encrypted message Y a cryptoanalyst must solve an excessively difficult equation to recover the original input X.

Example D:

Using data from our simple one element one input/output training Example A.

Known to the attacker: Encrypt key (E_{encr}) and encrypted code.

$$E_{encr} = \begin{bmatrix} 77 \\ 1056 \\ 501 \\ 900 \end{bmatrix}$$

encrypted data=3137_{16}

To solve for the original data, he must solve

$$77x_1 + 1056x_2 + 501x_3 + 900x_4 = 3137_{16}$$

with x_i being 4 bit,

which is one equation and four unknown. □

The above example is done on a simple single processing element model with a simple linear function. Given a multilayer network such as the back–propagation type with non–linear processing elements, even if the attacker knows the network, the problem mathematically increases in difficulty since the number of elements grows and thus the amount of required factorizations grows.

Even with a simple one cell example, for a crude cryptoanalysis method, one must solve the equation by trying 2^{16} combination of inputs to break one character.

Using a crude equation for Table 4.4:
Time in seconds = $2^{Number\ of\ bits} loops (10^{-9} sec\ computer/loop)\ 1000 computers$

Number of input bits per x_i	Time
4 (this report's element)	0.07 ns
8	4.3 ms
16	213 days
32	1.08×10^{17} centuries
64	3.67×10^{55} centuries

TABLE 4.4: EXHAUSTIVE SEARCH CRYPTOANALYSIS TIME FOR A SINGLE CELL

On the average it will take less then all combinations as it is probable that the solution can come anywhere in the search. An exhaustive search of 2^{16} loops for 2^{16} combinations poses little problem with the power of the computer but let's say one increases the same simple single layer input and output to a 32 -bit, 64-bit , 128-bit, or 256-bit input. Herein lies the basis behind the security of this system: a large range for the input of the network. Whereas up until now, only 16–bit inputs were used in a simple example, when this range is increased to 256 bit, the difficulty of working with such a large number renders any cryptoanalysis infeasible. Using an exhaustive search, Table 4.4 shows the amount of total possible time it would take to break one character given 1000 computers operating at 1 ns per loop operation (a very generous, fast time).

As with all cryptosystems, the time above can be minimized further if the system is susceptible to the problem of predictable frequency in the vocabulary. Namely, when the number representing trends such as 'the', 'a', space, double letters 'll', 'nn' exists, estimation of those characters are made easier. With this system, there exists a countermeasure in that one could use numbers not mapped to inject noise into the transmission thus breaking up any patterns. Here, since only 45 numbers are needed to represent 45 characters, there are $2^{16} - 45$ random numbers left to be used by some algorithm which would insert them into common words such as those mentioned above. This possibility was shown earlier in the in–depth example of Section C.

With the multi–element structure of the back–propagation network, the cryptoanalysis problem is exponentially greater with increase in number of network elements. Undoubtedly, the insurmountable time can be decreased given the luck factor in the probabilities and in due time further development in mathematics can solve in feasible time the NP complete problem. Nevertheless, at this date, the postulate is made that this is a very safe public–key cryptosystem.

2. Justification of the "Pseudo" Prefix

Ironically, the restrictions which necessitate the prefix "pseudo" for the system arise from the same attributes that make the system safe. Given a range of bits of input x, one cannot use all the possible combinations to train the network. For example, if each x was 64 bits long, one faces $2^{4\times64} = 2^{256}$ possible combinations. In order to encrypt anything between 0 and 2^{256}, all 2^{256} numbers must be matched to a unique y and trained to the network. This is comparable to the problem of the cryptoanalyst; it would take trillions of milleniums – not feasible.

The solution to this problem is avoidance. One needs only to train a certain range of number corresponding to the number of characters needed to be encrypted. For the NTP character set in this proposed system, one needs only a range of 45 out of numbers 2^{16} possible. However, both the encrypter and decrypter must know this range. How is this range to be kept a secret and still be passed to both parties? In order to make this neural network completely public–key, another PKS system is required to pass this range. It is suggested that the already popular Rivest Shamir Adleman (RSA) PKS system mentioned in Chapter II and III be used to pass this range.

In summary, key management involves the direct public disclosure of the encryption weights and the indirect public disclosure of the range of inputs via the RSA system. This leads to the question of why not use RSA completely and not be bothered with the neural network. The answer is that RSA is traditionally slower compared to neural networks (after training) and since the range of numbers used in encryption/decryption needs to be exchange only once prior to utilizing the system, one can afford to use RSA whereas for text encryption, a drawn–out repetitive real–time process, a neural network is much more efficient [Ref 12, 24].

E. PROBLEMS OF A NEURAL NETWORK AS A CRYPTOSYSTEM AND PROPOSED SOLUTIONS

The two potentially detrimental problems with the neural network scheme are that of the network weights not converging to an acceptable error for some non–linear training sets (non–convergence) and the mapping not guaranteed to be one to one (aliasing). Fortunately, the intrinsic versatility of neural networks is such that solutions to these problems exist.

The more serious of the two problems, non–convergence, can be easily illustrated by referring back to the one processing cell, one input/output training set example. With simply one cell, an addition of a second input/ouput pair – if not linearly related to the first pair – can cause the cell weights not to converge to acceptable errors; namely, there are no possible set of weights which will accommodate the correct outputs for both inputs. For example, the input/ouput pair $[2\ 1\ B\ 6]_{16}$ and $[0\ E\ F\ 3]_{16}$ is added to example 4.A. Using the old convergence weight for the original input/output, the actual output of the second pair is:

$$2(77) + 1(1056) + 11(501) + 6(900) = 12,121 = 2E59_{16}.$$

Clearly this is not the desired output for the second input. Hence, if one was to use the two data set above to train the single cell, the weights would not converge. One is then left with some restriction as to how to choose training set (mapping function). This restriction, can be easily exploited by a cryptoanalyst to break the system as he or she now knows that only certain mapping function is possible given knowledge of the system. Luckily, this restriction can be lifted with the back-propagation network used in this research.

As previously mentioned in Section A, a back-propagation network is an excellent mapping method of non-linear functions. Relying on this property, the training sets for encryption and decryption do not need to be linearly related. The more cells one adds to the network, the more non-linear functions can be mapped. Theoretically, with enough cells per layers, the weights will converge to acceptable errors given just any training data [Ref 24]. For the non-convergence example above, indeed the back-propagation network did prove to be the solution.

Additionally for public-key cryptography, one must bear in mind that the training data for encryption and decryption are related. For it to work, the weights of both encryption and decryption networks must converge. A training set that converges for encryption but its inverse training set does not yield converged weights for the decryption network is otherwise of no use in cryptography. From experimental data of the proposed 45 character encryption/decryption scheme, using the back-propagation system, problems of convergence were sometimes encountered. The reader is referred back to the experimental Section B where it was shown that when non-convergence does surface, the solution is to add more cells.

Apart from non-convergence, the second problem, aliasing, proved less serious but still needed to be dealt with. Aliasing occurs when, given a converged weights, two or more sets of inputs map to the same output. This nuisance can be attributed to the same problem which necessitated the "pseudo" prefix. Since one trains only a range of inputs within the vast possibility ($> 2^{256}$), the unused inputs could by chance map to one of the same chosen outputs.

Example E:

Again reverting back to the one cell, one input/output training set of Example A in Section A, an input of $[1\ 2\ A\ 6]_{16}$ along with encryption weights of [77 1056 501 900] yielded an encrypted code of $12599 = 3137_{16}$.

Let's use an input of $[7\ 1\ 4\ A]_{16}$ and the same converged weights. The encrypted code for this input will be

$$7(77) + 1(1056) + 4(501) + 10(900) = 12599 = 3137_{16},$$

which is the same output with the original input; hence aliasing has occured. □

Clearly aliasing is a theoretical possibility and thus a problem; however, in reality it can be easily be avoided by making sure one uses only the *trained* input/output pairs for encryption and decryption. This way, one knows exactly that a given encryption output should map back to the desired encryption input during decryption and not the aliased value. In fact, the alias problem can be exploited to the system's advantage. If certain aliasing problems are adapted intentionally, cryptoanalysis becomes more difficult. As previously explained in the "pseudo" justification section, only the desired parties knows the range of inputs to use whereas others do not. It is essential only to choose exact one–to–one mapping pairs in this range to avoid aliasing. Outside this range, any other inputs can have the aliasing effect, an actual benefit in extra safety.

F. DEVELOPMENT OF A COMPLETE BLOCK–DIAGRAM–LEVEL HARDWARE SCHEME USING A NEURAL NETWORK IN PKS

Up until now, most of the basic building blocks of a PKS using neural network have been discussed. Gathering all the essential blocks together, a possible block diagram proposal for an entire cryptosystem is shown in Figure 4.4.

Block by block description of Figure 4.4.

- The only component not yet delved into is the automatic generator of training input/ouput sets. This function can be fulfilled by a linear feedback shift register (LFSR). Given an input polynomial, it is a simple circuit capable of generating a random set of different numbers given. For this study, an LFSR of order 16 is necessary to generate $2^{16} - 1$ random numbers for both input/output pairs of encryption. For further insights on LFSR's, consult reference 28. After the input/ouput training sets of encryption is established by the LFSR, the decryption input/ouput training sets must be the inverse; namely ouput and input of encryption become input and input of decryption, respectively.

- Decrypt/encrypt neural net– Both networks are of the back-propagation type composed of 4 inputs , 1 ouput, 3 hidden layers with 25 elements per layer.

- Input Range Exchange– As discussed in Section D.2, the RSA hardware of Chapter III can be used to send the range thus making this a "pseudo" PKS.

- Network Weights– The weights of the neural networks must be able to undergo changes during training and then be set to constants once the the converged weights are obtained via training or received from opposite parties. Simple latches and switches seem adequate for the task although no detail studies are made.

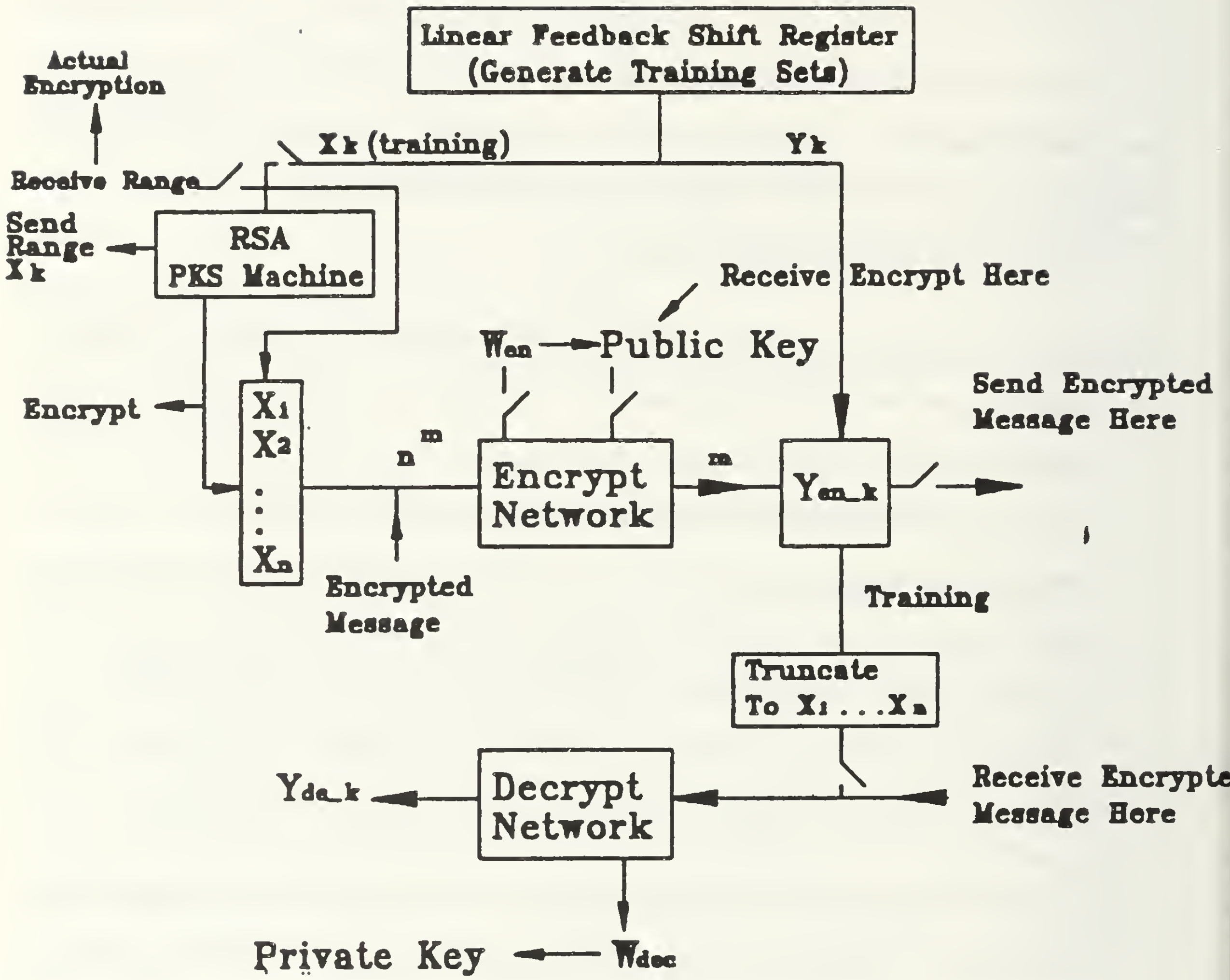

Figure 4.4: Neural Network in PKS

A working model of a public-key cryptosystem based on neural networks has been designed. It is merely a sample model which can be applied in limited usage; however, the idea behind the system deserves recognition as a worthwhile alternative to PKS.

V. CONCLUSION

This thesis has presented some novel approaches to public-key cryptosystems. The focus was centered on a specific hardware implementation and a completely new angle to PKS using neural networks. In both issues, research produced working models when simulated by computers.

The hardware implementation for a modulo reduction unit in a fast exponentiator – an essential device in the most popular PKS, RSA cryptosystem – was developed based on the sum–of–residues method (SOR). The design is based on the concept of modularity. The modular unit can be conveniently connected to form a fast exponentiator for numbers of any length. The result is a working VLSI layout when simulated by RNL (Appendix C). The efficiency in speed and size, though offered in the study, remains issues to be considered when the unit is to be used in real–world applications. If the speed and size given hereby are acceptable to a certain application then this unit is perhaps a viable alternative to existing technology due to its advantage in modularity.

The second part of this thesis involves the use of neural networks in PKS. To the author's knowledge, the attempt to integrate neural networks into cryptography is a novel idea. Whether it is either original or even revolutionary remains to be seen. That the goal is at all plausible is an unanticipated surprise when the experimental results confirmed it so. This is not to say that plausibility means practicality. So far, all that is proven is that the concept *works*. Whether the scheme is *feasible* needs further research.

From data gathered in Tables 2.4 and 4.4, one can conclude that at 256 bit in length for the key in RSA and input in the neural network-based cryptosystem, exhaustive cryptoanalysis faces infeasible time limit. For all practical purpose, requiring trillion of milleniums to break, the system of this thesis is as safe as any current PKS (Table 4.4). Additionally, the most significant advantage in using neural networks in PKS is that there is no need for fast exponentiation which has proven to be slow for large exponents and modulus [Ref 2]. The only necessary operations in a back–propagation network are multiplication, addition and hyperbolic tangent (or other non–linear functions.) The computational feasibility of the neural network scheme, however, is not explored here and is left to follow–on research.

At present, the example system only applies for input ranging 16 bit in length. For the system to be secured, it is suggested that the range be extended to 256 bit. Intuitively, if one single network is to be used to map numbers with 256 bit range, it will have to be large and thus will slow down the system. However, if parallel processing is available and one can afford to design a 256 bit cryptosystem based on 16 16-bit neural networks, the results of this paper will be of value. Furthermore, only the back–propagation network was used in this research. Given the multitudes of network types in various applications, there may exist other schemes capable of using other networks.

This paper is intended to pioneer the idea of neural network in cryptosystem. As such it claims only the initiative in a novel avenue to cryptography. The proposed theory of employing neural networks in cryptography now ends with a call for further research into the efficiency, speed and possibilities of more capable networks. The key to the knowledge gathered so far is that a new method is postulated and there seems to be some merit in that it works with some restrictions. These restrictions may be lifted by further investigation or perhaps there shall come a disproval which may destroy the entire scheme altogether. Be that as it may, time constraint dictates that this introductory study terminates with many aspirations of fueling follow–on research in this subject.

APPENDIX A
SUPPLEMENTARY PROGRAMS

The following programs are provided to supplement background knowledge in public–key cryptography. In order, they are: fast exponentiation, greatest common divisor, inverse, and factorization. The first three programs are written in C [Ref 2] and run on Unix while factorization is in Matlab code and ran on an IBM '486, 50MHz, 16MB.

```
*******************************************************************

/*
This program uses the fast exponential algorithm to compute the operation:
a^z mod n. It is intended as an example of software implementation of the
RSA public key cryptosystem. */

#include <stdio.h>

/* The algorithm is contained in the following function to be called when
necessary. */

int fastexp(a, z, n)
int a, z, n;
{
   int x = 1;
   while (z)
```

```
{
     while (!(z % 2))
     {
       z /= 2;
       a = ((a%n)*(a % n)) %n;
     }
     z--;
     x = ((x % n)*(a % n)) % n;
}
   return (x);
}
main()
{
   int a, z, n, t;
   printf("a^z(mod n). Enter a, z, n ");
   scanf("%d %d %d0",&a,&z,&n);
   t= fastexp( a, z, n);
   printf("Result = %d\n", t );
}
```

****

/*

This program uses Euclid's algorithm to solve for the greatest common denominator (gcd) of two number. Given two input integers, a and n, this program provides their mutual gcd. This is intended to be an example for

```
generating keys in the RSA public key system */

#include <stdio.h>
main()
{
   int   g[100];      /* Initialize an array for gcd */
   int   i=1;
   printf ("gcd of a,n. Enter a,n separated by space:");
   scanf ("%d %d0", &g[0], &g[1]);
   while (g[i])
   {
     g[i+1] = g[i-1] % g[i];
     i++;
   }
   printf ("gcd of %d and %d is %d \n",g[0],g[1],g[i-1]);
}
```

****

```
/* This program compute the inverse, x, of a and n (0<a<n) such that
ax (mod n) = 1 */

#include <stdio.h>
main()
{
   int g[100], u[100], v[100];   /* Initialize arrays for indexing */
```

```
    int i=1;                        /* Beginning index # of loop */
    int y,n,a;                   /* Defining input and intermediate var. */
    printf ("inverse of a,n. Enter a,n separated by space: ");
    scanf  ("%d %d0", &a, &n);    /* Read in a and n */
    g[0]= n;
    g[1]= a;
    u[0]= v[1] = 1;
    u[1] = v[0] = 0;
    while (g[i])
    {
       g[i]= u[i] * n + v[i] * a;
       y= g[i-1]/g[i];
       g[i+1] = g[i-1] - y*g[i];
       u[i+1] = u[i-1] - y*u[i];
       v[i+1] = v[i-1] - y*v[i];
       i++;
    }                            /* Using extension of Euclid's gcd algo */
    if (v[i-1] <= 0)
    {
       printf ("inv of %d and %d is %d \n", a,n,v[i-1]+n);
    }
    else
    {
       printf ("inv of %d and %d is %d \n",a,n,v[i-1]+2*n);
    }
}
```

```
************************************************************************

% This is a Matlab program designed to factorize a product of two
% primes for the cryptoanalysis of the RSA public-key cryptosystem.
% Intended merely to show the futility of factorizing large numbers,
% it employs a naive exhaustive search method of dividing and
% checking the remainder of the division of the product and every
% possible odd numbers until a factor is found. To use the program,
% simply type rsafac('product of 2 primes').

function[x]=rsafac(z);                % Enter the product.
w=round(sqrt(z));    % Factor can not be larger than
   % the square root of the product.

for n=1:2:w    % No need to test even numbers, and
   % limit of search is w.
   v=z/n;    % Testing by dividing products by
   % odd numbers.
  if (rem(v,1)==0)    % If v is integer then
    x=[n,v];    % n and v are factors.
    n=w;    % Exit loop once factors are found
  end
end
```

APPENDIX B

RNL SIMULATION OF MODULO REDUCTION UNIT

The following examples are indicative of the successful RNL simulation [Ref 3] of the final modulo reduction unit. The unit simulated here is limited to modulo numbes of 4–bit length. The RNL control file, stimulation file for one example are included along with simulation results of 5 modulo operations.

Sample control file for RNL simulation of 5 mod 7 using modulo reduction layout of Figure 3.11.

```
; The name of this control file for rnl is: mod1.l
; Simulation for modulo reduction unit of Chapter 3.
; LOAD STANDARD LIBRARY ROUTINES
(load "uwstd.l")
(load "uwsim.l")
; FILE WHICH WILL LOG THE RESULTS
(log-file "mod1.rlog")
; READ IN THE BINARY NETWORK FILE
(read-network "mod1")
; DEFINE THE TIME SCALE FOR SIMULATION
(setq incr 90)
; DEFINE INPUT VECTOR IF ANY, standard STYLE
(defvec '(bit state s3 s2 s1 s0 ))
```

```
;  DEFINE INPUT VECTOR IF ANY, SINGLE INDEX STYLE

;  DEFINE INPUT VECTOR IF ANY, double index STYLE

;  STANDARD REPORT FORMAT DEFINITION.

(def-report '("response= " cl1 cl2 in i3 i2 i1 (vec state)))

;  PLOTFILE SPECIFIED

openplot "mod1.beh"

;  LOGIC ANALYZER STYLE OUTPUT FORMAT SELECTION.

(setq lanalyze t)

(wr-format)

;  GLITCH DETECTOR SELECTION.

(setq glitch-detect t)

;  NODE TRANSIENTS REPORT DEFINITION.

(chflag '( s3 s2 s1 s0))

;  TRIGGER CONDITION SET-UP

;  ADDITIONAL SIMULATION SET-UP COMMAND LINES.

(printf "Commence simulation...\n")

;  SPECIFICATION OF A TIME/BASENAME FILE FOR INCLUSION.

(load "mod1.time")

;  ADDITIONAL WRAP-UP COMMAND LINES.

(printf "...completed simulation!\n")

exit

;  GEN-CONTROL COMPLETED.

**************************************************************

;The following is the stimulation file for the input to the rnl simulation

;above for 5 mod 7.

Sample <  >.stim file for 5 mod 7:
```

```
time_range 0 10
in 0 h 0 1 2 h 4          ; Note 101 is entered for 5
inn 0 1 0 h 2 1 4         ; Simply inverse of in

cl1 2 1 0 h 1             ; 2-phase clocks
cl1n 2 h 0 1 1

cl2 2 h 0 1 1
cl2n 2 1 0 h 1

opt 0 h 0 x 1             ; Initializing MUX select
optn 0 1 0 x 1

m0 0 h 0                  ; 2's complement of 7 is 1001
m1 0 1 0                  ; Modulo number inputs
m2 0 1 0
m3 0 h 0

s3 0 1 0 x 1              ; Initializing summer
s2 0 1 0 x 1
s1 0 1 0 x 1
s0 0 1 0 x 1

i3 0 1 0 x 1              ; Initializing 1st residue to 1
i2 0 1 0 x 1
i1 0 h 0 x 1
```

```
report 1 0

*************************************************************

;The following is the RNL simulation result of stimulation file above
;5 mod 7 :
; 118 nodes, transistors: enh=68 intrinsic=0 p-chan=56 dep=0
;low-power=0 pullup=0 resistor=0
; Report format of logic analyzer style output
 time  cl1 cl2    in    i3 i2 i1    state(result)
                  *     **
Commence simulation...
9      0   1      1     0 0 1       0000
18     1   0      1     0 0 1       0001 - 1st clk pulse
27     0   1      0     0 1 0       0001
36     1   0      0     0 1 0       0001 - 2nd clk pulse
45     0   1      1     1 0 0       0001
54     1   0      1     1 0 0       0101 - 3rd clk pulse ***
63     0   1      1     0 0 1       0101
...completed simulation!
* Input is 101= 5 (Note input taken at each rising clock edge.)
** Residues are 1,2,4,1,2,4... for mod 7.
*** 5 mod 7 = 0101= 5.

***********************************************************************
```

```
;The following is a second RNL simulation result (10 mod 6):
; 118 nodes, transistors: enh=68 intrinsic=0 p-chan=56
;dep=0 low-power=0 pullup=0 resistor=0
; Report format of logic analyzer style output
 time  cl1 cl2    in    i3 i2 i1    state(result)
                  *     **
Commence simulation...
9       0   1     0     0 0 1       0000
18      1   0     0     0 0 1       0000 - 1st clk pulse
27      0   1     1     0 1 0       0000
36      1   0     1     0 1 0       0010 - 2nd clk pulse
45      0   1     0     1 0 0       0010
54      1   0     0     1 0 0       0010 - 3rd clk pulse
63      0   1     1     0 1 0       0010
72      1   0     1     0 1 0       0100 - 4th clk pulse ***
...completed simulation!
* Input is 1010= 10.
** Residues are 1,2,4... for mod 7.
*** 10 mod 6 = 0100= 6.
```

```
*******************************************************************
```

```
;Third RNL simulation using 10 mod 7:
; 118 nodes, transistors: enh=68 intrinsic=0 p-chan=56
; dep=0 low-power=0 pullup=0 resistor=0
```

```
; Report format of logic analyzer style output
 time  cl1 cl2    in    i3 i2 i1    state(result)
                  *     **
Commence simulation...
9       0   1     0     0 0 1       0000
18      1   0     0     0 0 1       0000 - 1st clk pulse
27      0   1     1     0 1 0       0000
36      1   0     1     0 1 0       0010 - 2nd clk pulse
45      0   1     0     1 0 0       0010
54      1   0     0     1 0 0       0010 - 3rd clk pulse
63      0   1     1     0 0 1       0010
72      1   0     1     0 0 1       0011 - 4th clk pulse ***
...completed simulation!
* Input is 1010= 10.
** Residues for mod 7 is 1,2,4,1,2,4...
**10 mod 7= 0011 = 3.

*****************************************************************

; Fourth RNL simulation using 11 mod 6.
; 118 nodes, transistors: enh=68 intrinsic=0 p-chan=56
; dep=0 low-power=0 pullup=0 resistor=0
; Report format of logic analyzer style output
 time  cl1 cl2    in    i3 i2 i1    state(result)
                  *     **
Commence simulation...
```

```
9       0   1      1     0 0 1       0000
18      1   0      1     0 0 1       0001 - 1st clk pulse
27      0   1      1     0 1 0       0001
36      1   0      1     0 1 0       0011 - 2nd clk pulse
45      0   1      0     1 0 0       0011
54      1   0      0     1 0 0       0011 - 3rd clk pulse
63      0   1      1     0 1 0       0011
72      1   0      1     0 1 0       0101 - 4th clk pulse***
81      0   1      1     1 0 0       0101
...completed simulation!

* input is 1011= 11.
** Residues of mod 6 are 1,2,4,2,4...
*** 11 mod 6= 0101= 5

************************************************************

; Fifth RNL simulation with 17 mod 5
; 118 nodes, transistors: enh=68 intrinsic=0 p-chan=56
; dep=0 low-power=0 pullup=0 resistor=0
; Report format of logic analyzer style output
 time  cl1 cl2    in    i3 i2 i1    state(result)
                  *     **
Commence simulation...
9       0   1      1     0 0 1       0000
18      1   0      1     0 0 1       0001 - 1st clk pulse
```

```
27      0   1       0      0 1 0        0001
36      1   0       0      0 1 0        0001 - 2nd clk pulse
45      0   1       0      1 0 0        0001
54      1   0       0      1 0 0        0001 - 3rd clk pulse
63      0   1       0      0 1 1        0001
72      1   0       0      0 1 1        0001 - 4th clk pulse
81      0   1       1      0 0 1        0001
90      1   0       1      0 0 1        0010 - 5th clk pulse***
99      0   1       1      0 1 0        0010
...completed simulation!
```

* Input is 10001= 17.

** Residues of mod 5 are 1,2,4,3,1,2,4,3...

*** 17 mod 5=0010 = 2.

APPENDIX C

SAMPLE NEURAL NETWORK FROM NEURALWARE

The following is data for the encryption and decryption neural network used in Chapter IV in-depth example. The network data is formatted from Neuralware [Ref 26] "annotated" option once convergence is reached. This option provides all the necessary parameters to reconstruct the network trained by data from Table 4.1. Of the many parameters, those of interest are learning iterations (375642 for encryption and 333877 for decryption), error function (standard ≡ hyperbolic tangent), learning rule (delta-rule), and the processing elements' data. Of the element's data, the error for each element's output was approximately zero once convergence is reached. The weight data are not included other than the number of weights going to each element. The reason for this omission is that it is not pertinent. With the data offered here and Table 4.1, one can reconstruct the encryption and decryption network using Neuralware.

```
Title: Encryption Network for In--Depth Example
      Display Mode: Network                   Type: Hetero-Associative
   Control Strategy: backprop                L/R Schedule: backprop
     375642 Learn          0 Recall            0 Layer
         16 Aux 1          0 Aux 2             0 Aux 3
L/R Schedule: backprop
   Recall Step                 1         0         0         0         0
     Firing Density     100.0000    0.0000    0.0000    0.0000    0.0000
     Gain                 1.0000    0.0000    0.0000    0.0000    0.0000
     Gain                 1.0000    0.0000    0.0000    0.0000    0.0000
   Learn Step               5000    0         0         0         0
     Coefficient 1        0.9000    0.0000    0.0000    0.0000    0.0000
     Coefficient 2        0.6000    0.0000    0.0000    0.0000    0.0000
     Coefficient 3        0.0000    0.0000    0.0000    0.0000    0.0000
IO Parameters
         Learn Data: File Rand. (Encryption file here) Binary
        Recall Data: File Seq. (Encryption file here)
```

```
       Result File: Desired Output, Output
    UserIO Program: userio
        I/P Ranges:    -1.0000,     1.0000
        O/P Ranges:    -0.8000,     0.8000
     I/P Start Col:          1              MinMax Table: sama
     O/P Start Col:          5         Number of Entries: 5
MinMax Table <sama>:
Col:           1           2           3           4           5
Min:      0.0000      1.0000      1.0000      1.0000   2445.0000
Max:          15          11          12          14   6.28e+004
Layer: 1
        PEs: 1        Wgt Fields: 2                     Sum: Sum
    Spacing: 5         F' offset: 0.00             Transfer: Linear
      Shape: Square                                  Output: Direct
      Scale: 1.00      Low Limit: -9999.00      Error Func: standard
     Offset: 0.00     High Limit: 9999.00            Learn: --None--
   Init Low: -0.100    Init High: 0.100    L/R Schedule: (Network)
   Winner 1: None                                Winner 2: None
   PE: Bias
       1.000 Err Factor       0.000 Desired
       0.000 Sum              1.000 Transfer          1.000 Output
    0 Weights      -291.920 Error            0.000 Current Error
Layer: In
        PEs: 4        Wgt Fields: 1                     Sum: Sum
    Spacing: 5         F' offset: 0.00             Transfer: Linear
      Shape: Square                                  Output: Direct
    Scale: 1.00      Low Limit: -9999.00        Error Func: standard
   Offset: 0.00     High Limit: 9999.00              Learn: --None--
Init Low: -0.100   Init High: 0.100       L/R Schedule: (Network)
   Winner 1: None                                Winner 2: None
   PE: 2
       1.000 Err Factor      -0.867 Desired
      -0.867 Sum             -0.867 Transfer         -0.867 Output
*** 0 Weights         0.000 Error            0.000 Current Error
*** From here on all error for all PE's are 0's.
   PE: 3
       1.000 Err Factor      -0.800 Desired
      -0.800 Sum             -0.800 Transfer         -0.800 Output
   PE: 4
       1.000 Err Factor       0.636 Desired
       0.636 Sum              0.636 Transfer          0.636 Output
   PE: 5
       1.000 Err Factor       0.692 Desired
       0.692 Sum              0.692 Transfer          0.692 Output
Layer: Hidden1
        PEs: 25       Wgt Fields: 2                     Sum: Sum
    Spacing: 5         F' offset: 0.00             Transfer: TanH
      Shape: Square                                  Output: Direct
    Scale: 1.00      Low Limit: -9999.00        Error Func: standard
 Offset: 0.00     High Limit: 9999.00            Learn: Delta-Rule
 Init Low: -0.100   Init High: 0.100      L/R Schedule: hidden1
   Winner 1: None                                Winner 2: None
L/R Schedule: hidden1
   Recall Step               1         0         0         0         0
     Firing Density  100.0000    0.0000    0.0000    0.0000    0.0000
     Gain                1.0000    0.0000    0.0000    0.0000    0.0000
```

```
  Gain                  1.0000    0.0000    0.0000    0.0000    0.0000
 Learn Step             10000     30000     70000     150000    310000
  Coefficient 1         0.3000    0.1800    0.0648    0.0084    0.0001
  Coefficient 2         0.3000    0.1800    0.0648    0.0084    0.0001
  Coefficient 3         0.1000    0.1000    0.1000    0.1000    0.1000
 PE: 6
     1.000 Err Factor      0.000 Desired
     0.044 Sum             0.044 Transfer           0.044 Output
*** 5 Weights         0.000 Error              0.000 Current Error
***   From here on all weights are 5 and errors are 0.
 PE: 7
     1.000 Err Factor      0.000 Desired
     0.612 Sum             0.546 Transfer           0.546 Output
 PE: 8
     1.000 Err Factor      0.000 Desired
    -0.123 Sum            -0.123 Transfer          -0.123 Output
 PE: 9
     1.000 Err Factor      0.000 Desired
     0.500 Sum             0.462 Transfer           0.462 Output
 PE: 10
     1.000 Err Factor      0.000 Desired
    -1.634 Sum            -0.927 Transfer          -0.927 Output
 PE: 11
     1.000 Err Factor      0.000 Desired
    -0.069 Sum            -0.069 Transfer          -0.069 Output
 PE: 12
     1.000 Err Factor      0.000 Desired
     0.145 Sum             0.144 Transfer           0.144 Output
 PE: 13
     1.000 Err Factor      0.000 Desired
    -0.008 Sum            -0.008 Transfer          -0.008 Output
 PE: 14
     1.000 Err Factor      0.000 Desired
    -0.305 Sum            -0.296 Transfer          -0.296 Output
 PE: 15
     1.000 Err Factor      0.000 Desired
    -0.045 Sum            -0.045 Transfer          -0.045 Output
 PE: 16
     1.000 Err Factor      0.000 Desired
    -0.376 Sum            -0.359 Transfer          -0.359 Output
 PE: 17
     1.000 Err Factor      0.000 Desired
    -0.037 Sum            -0.037 Transfer          -0.037 Output
 PE: 18
     1.000 Err Factor      0.000 Desired
    -2.242 Sum            -0.978 Transfer          -0.978 Output
 PE: 19
     1.000 Err Factor      0.000 Desired
     0.023 Sum             0.023 Transfer           0.023 Output
 PE: 20
     1.000 Err Factor      0.000 Desired
     0.228 Sum             0.224 Transfer           0.224 Output
 PE: 21
     1.000 Err Factor      0.000 Desired
    -2.312 Sum            -0.981 Transfer          -0.981 Output
 PE: 22
```

```
        1.000 Err Factor       0.000 Desired
        1.274 Sum              0.855 Transfer          0.855 Output
    PE: 23
        1.000 Err Factor       0.000 Desired
        0.031 Sum              0.031 Transfer          0.031 Output
    PE: 24
        1.000 Err Factor       0.000 Desired
        0.029 Sum              0.029 Transfer          0.029 Output
    PE: 25
        1.000 Err Factor       0.000 Desired
        0.816 Sum              0.673 Transfer          0.673 Output
    PE: 26
        1.000 Err Factor       0.000 Desired
       -0.286 Sum             -0.279 Transfer         -0.279 Output
    PE: 27
        1.000 Err Factor       0.000 Desired
       -0.299 Sum             -0.290 Transfer         -0.290 Output
    PE: 28
        1.000 Err Factor       0.000 Desired
        1.650 Sum              0.929 Transfer          0.929 Output
    PE: 29
        1.000 Err Factor       0.000 Desired
        0.891 Sum              0.712 Transfer          0.712 Output
    PE: 30
        1.000 Err Factor       0.000 Desired
        0.440 Sum              0.414 Transfer          0.414 Output
Layer: Hidden2
         PEs: 25      Wgt Fields: 2                     Sum: Sum
     Spacing: 5        F' offset: 0.00             Transfer: TanH
       Shape: Square                                 Output: Direct
     Scale: 1.00     Low Limit: -9999.00        Error Func: standard
  Offset: 0.00    High Limit: 9999.00                Learn: Delta-Rule
   Init Low: -0.100   Init High: 0.100        L/R Schedule: hidden2
    Winner 1: None                                 Winner 2: None
L/R Schedule: hidden2
    Recall Step                1         0         0         0         0
      Firing Density    100.0000    0.0000    0.0000    0.0000    0.0000
      Gain                1.0000    0.0000    0.0000    0.0000    0.0000
      Gain                1.0000    0.0000    0.0000    0.0000    0.0000
    Learn Step             10000     30000     70000    150000    310000
      Coefficient 1       0.2500    0.1500    0.0540    0.0070    0.0001
      Coefficient 2       0.3000    0.1800    0.0648    0.0084    0.0001
      Coefficient 3       0.1000    0.1000    0.1000    0.1000    0.1000
    PE: 31
        1.000 Err Factor       0.000 Desired
        0.221 Sum              0.218 Transfer          0.218 Output
***26 Weights        -0.000 Error              -0.000 Current Error
***  From here on all PE's have 26 weights, approximately 0 error.
    PE: 32
        1.000 Err Factor       0.000 Desired
       -1.459 Sum             -0.897 Transfer         -0.897 Output
    PE: 33
        1.000 Err Factor       0.000 Desired
       -2.230 Sum             -0.977 Transfer         -0.977 Output
    PE: 34
        1.000 Err Factor       0.000 Desired
```

```
    -0.297 Sum              -0.288 Transfer          -0.288 Output
PE: 35
     1.000 Err Factor        0.000 Desired
    -0.168 Sum              -0.167 Transfer          -0.167 Output
PE: 36
     1.000 Err Factor        0.000 Desired
     0.315 Sum               0.305 Transfer           0.305 Output
PE: 37
     1.000 Err Factor        0.000 Desired
     1.152 Sum               0.818 Transfer           0.818 Output
PE: 38
     1.000 Err Factor        0.000 Desired
    -0.165 Sum              -0.164 Transfer          -0.164 Output
PE: 39
     1.000 Err Factor        0.000 Desired
    -1.256 Sum              -0.850 Transfer          -0.850 Output
PE: 40
     1.000 Err Factor        0.000 Desired
    -0.520 Sum              -0.477 Transfer          -0.477 Output
PE: 41
     1.000 Err Factor        0.000 Desired
    -1.282 Sum              -0.857 Transfer          -0.857 Output
PE: 42
     1.000 Err Factor        0.000 Desired
     2.801 Sum               0.993 Transfer           0.993 Output
PE: 43
     1.000 Err Factor        0.000 Desired
     0.082 Sum               0.081 Transfer           0.081 Output
PE: 44
     1.000 Err Factor        0.000 Desired
    -2.658 Sum              -0.990 Transfer          -0.990 Output
PE: 45
     1.000 Err Factor        0.000 Desired
     4.263 Sum               1.000 Transfer           1.000 Output
PE: 46
     1.000 Err Factor        0.000 Desired
    -0.159 Sum              -0.158 Transfer          -0.158 Output
PE: 47
     1.000 Err Factor        0.000 Desired
    -0.068 Sum              -0.068 Transfer          -0.068 Output
PE: 48
     1.000 Err Factor        0.000 Desired
    -0.707 Sum              -0.609 Transfer          -0.609 Output
PE: 49
     1.000 Err Factor        0.000 Desired
    -0.527 Sum              -0.483 Transfer          -0.483 Output
PE: 50
     1.000 Err Factor        0.000 Desired
    -3.316 Sum              -0.997 Transfer          -0.997 Output
PE: 51
     1.000 Err Factor        0.000 Desired
    -1.019 Sum              -0.770 Transfer          -0.770 Output
PE: 52
     1.000 Err Factor        0.000 Desired
     0.934 Sum               0.733 Transfer           0.733 Output
PE: 53
```

```
      1.000 Err Factor      0.000 Desired
     -0.033 Sum            -0.033 Transfer          -0.033 Output
   PE: 54
      1.000 Err Factor      0.000 Desired
     -2.768 Sum            -0.992 Transfer          -0.992 Output
   PE: 55
      1.000 Err Factor      0.000 Desired
      0.017 Sum             0.017 Transfer           0.017 Output
Layer: Hidden3
        PEs: 25       Wgt Fields: 2                      Sum: Sum
    Spacing: 5         F' offset: 0.00              Transfer: TanH
      Shape: Square                                   Output: Direct
    Scale: 1.00     Low Limit: -9999.00         Error Func: standard
 Offset: 0.00    High Limit: 9999.00                Learn: Delta-Rule
  Init Low: -0.100   Init High: 0.100        L/R Schedule: hidden3
   Winner 1: None                                 Winner 2: None
L/R Schedule: hidden3
   Recall Step                1         0         0         0         0
     Firing Density     00.0000    0.0000    0.0000    0.0000    0.0000
     Temperature         0.0000    0.0000    0.0000    0.0000    0.0000
     Gain                1.0000    0.0000    0.0000    0.0000    0.0000
     Gain                1.0000    0.0000    0.0000    0.0000    0.0000
   Learn Step             10000     30000     70000    150000    310000
     Coefficient 1       0.2000    0.1200    0.0432    0.0056    0.0001
     Coefficient 2       0.3000    0.1800    0.0648    0.0084    0.0001
     Coefficient 3       0.1000    0.1000    0.1000    0.1000    0.1000
   PE: 56
      1.000 Err Factor      0.000 Desired
      0.421 Sum             0.398 Transfer           0.398 Output
   PE: 57
      1.000 Err Factor      0.000 Desired
     -0.212 Sum            -0.209 Transfer          -0.209 Output
   PE: 58
      1.000 Err Factor      0.000 Desired
      0.145 Sum             0.144 Transfer           0.144 Output
   PE: 59
      1.000 Err Factor      0.000 Desired
     -0.139 Sum            -0.138 Transfer          -0.138 Output
   PE: 60
      1.000 Err Factor      0.000 Desired
     -0.209 Sum            -0.206 Transfer          -0.206 Output
   PE: 61
      1.000 Err Factor      0.000 Desired
      0.137 Sum             0.136 Transfer           0.136 Output
   PE: 62
      1.000 Err Factor      0.000 Desired
      0.151 Sum             0.150 Transfer           0.150 Output
   PE: 63
      1.000 Err Factor      0.000 Desired
     -0.306 Sum            -0.297 Transfer          -0.297 Output
   PE: 64
      1.000 Err Factor      0.000 Desired
      0.669 Sum             0.584 Transfer           0.584 Output
   PE: 65
      1.000 Err Factor      0.000 Desired
     -0.153 Sum            -0.152 Transfer          -0.152 Output
```

```
PE: 66
    1.000 Err Factor        0.000 Desired
   -0.436 Sum              -0.410 Transfer         -0.410 Output
PE: 67
    1.000 Err Factor        0.000 Desired
   -0.086 Sum              -0.086 Transfer         -0.086 Output
PE: 68
    1.000 Err Factor        0.000 Desired
    0.082 Sum               0.082 Transfer          0.082 Output
PE: 69
    1.000 Err Factor        0.000 Desired
   -0.108 Sum              -0.108 Transfer         -0.108 Output
PE: 70
    1.000 Err Factor        0.000 Desired
    0.071 Sum               0.071 Transfer          0.071 Output
PE: 71
    1.000 Err Factor        0.000 Desired
    0.181 Sum               0.179 Transfer          0.179 Output
PE: 72
    1.000 Err Factor        0.000 Desired
    0.233 Sum               0.229 Transfer          0.229 Output
PE: 73
    1.000 Err Factor        0.000 Desired
   -0.244 Sum              -0.239 Transfer         -0.239 Output
PE: 74
    1.000 Err Factor        0.000 Desired
    0.378 Sum               0.361 Transfer          0.361 Output
PE: 75
    1.000 Err Factor        0.000 Desired
   -0.318 Sum              -0.308 Transfer         -0.308 Output
PE: 76
    1.000 Err Factor        0.000 Desired
   -0.484 Sum              -0.449 Transfer         -0.449 Output
PE: 77
    1.000 Err Factor        0.000 Desired
    0.128 Sum               0.127 Transfer          0.127 Output
PE: 78
    1.000 Err Factor        0.000 Desired
   -0.047 Sum              -0.047 Transfer         -0.047 Output
PE: 79
    1.000 Err Factor        0.000 Desired
   -0.379 Sum              -0.361 Transfer         -0.361 Output
PE: 80
    1.000 Err Factor        0.000 Desired
    0.647 Sum               0.569 Transfer          0.569 Output
Layer: Out
        PEs: 1       Wgt Fields: 2                    Sum: Sum
    Spacing: 5        F' offset: 0.00            Transfer: TanH
     Shape: Square                                 Output: Direct
    Scale: 1.00     Low Limit: -9999.00       Error Func: standard
 Offset: 0.00    High Limit: 9999.00               Learn: Delta-Rule
   Init Low: -0.100   Init High: 0.100        L/R Schedule: out
   Winner 1: None                                Winner 2: None
L/R Schedule: out
   Recall Step            1        0        0        0        0
     Input Clamp      0.0000   0.0000   0.0000   0.0000   0.0000
```

```
   Firing Density  100.0000    0.0000    0.0000    0.0000    0.0000
   Temperature       0.0000    0.0000    0.0000    0.0000    0.0000
   Gain              1.0000    0.0000    0.0000    0.0000    0.0000
   Gain              1.0000    0.0000    0.0000    0.0000    0.0000
 Learn Step           10000     30000     70000    150000    310000
   Coefficient 1     0.1500    0.0900    0.0324    0.0042    0.0001
   Coefficient 2     0.3000    0.1800    0.0648    0.0084    0.0001
   Coefficient 3     0.1000    0.1000    0.1000    0.1000    0.1000
 PE: 81
     1.000 Err Factor      -0.525 Desired
    -0.583 Sum             -0.525 Transfer           -0.525 Output
 26 Weights         0.000 Error              0.000 Current Error
```

****

Resulting actual output and desired output for encryption after convergence in accordance with Table 4.1 input:

Desired:	Actual:
12828.000000	12827.522461
38939.000000	38939.464844
41525.000000	41524.664063
16986.000000	16985.642188
25907.000000	25907.292969
41305.000000	41304.957031
34609.000000	34609.128906
9880.000000	9880.100586
37175.000000	37175.384375
26139.000000	26138.814453
4942.000000	4942.453223
30022.000000	30021.833984
8523.000000	8523.165039
21386.000000	21385.605469
4780.000000	4779.714844
26946.000000	26946.346094
33050.000000	33050.152344
29325.000000	29324.822266
13751.000000	13750.862305
62803.000000	62803.332031
59988.000000	59987.847656
3018.000000	3017.878906
45353.000000	45353.355469
54086.000000	54086.285156
50536.000000	50536.437500
2445.000000	2445.414014
54097.000000	54097.246094
45926.000000	45926.305469

****

```
Title: Decryption Network for In--Depth Example of Chapter 4
       Display Mode: Network             Type: Hetero-Associative
      Display Style: default
   Control Strategy: backprop              L/R Schedule: backprop
      333877 Learn              0 Recall                0 Layer
          16 Aux 1              0 Aux 2                 0 Aux 3
L/R Schedule: backprop
   Recall Step               1         0         0         0         0
```

```
     Firing Density  100.0000   0.0000   0.0000   0.0000   0.0000
     Gain              1.0000   0.0000   0.0000   0.0000   0.0000
   Learn Step          5000          0        0        0        0
     Coefficient 1     0.9000   0.0000   0.0000   0.0000   0.0000
     Coefficient 2     0.6000   0.0000   0.0000   0.0000   0.0000
     Coefficient 3     0.0000   0.0000   0.0000   0.0000   0.0000
IO Parameters
         Learn Data: File Rand. (decryption file) Binary
        Recall Data: File Seq. (decryption)
        Result File: Desired Output, Output
     UserIO Program: userio
         I/P Ranges:    -1.0000,     1.0000
         O/P Ranges:    -0.8000,     0.8000
      I/P Start Col:          1              MinMax Table: samb
      O/P Start Col:          5         Number of Entries: 5
MinMax Table <samb>:
Col:            1           2           3           4          5
Min:       0.0000      1.0000      1.0000      1.0000  2445.0000
Max:           15          11          12          14  6.28e+004
Layer: 1
        PEs: 1         Wgt Fields: 2                      Sum: Sum
    Spacing: 5          F' offset: 0.00              Transfer: Linear
      Shape: Square                                    Output: Direct
    Scale: 1.00       Low Limit: -9999.00         Error Func: standard
   Offset: 0.00      High Limit: 9999.00               Learn: --None--
Init Low: -0.100     Init High: 0.100          L/R Schedule: (Network)
   Winner 1: None                                   Winner 2: None
   PE: Bias
       1.000 Err Factor       0.000 Desired
       0.000 Sum              1.000 Transfer            1.000 Output
    0 Weights        -247.657 Error              0.000 Current Error
Layer: In
        PEs: 4         Wgt Fields: 1                      Sum: Sum
    Spacing: 5          F' offset: 0.00              Transfer: Linear
      Shape: Square                                    Output: Direct
    Scale: 1.00       Low Limit: -9999.00         Error Func: standard
   Offset: 0.00      High Limit: 9999.00               Learn: --None--
Init Low: -0.100     Init High: 0.100          L/R Schedule: (Network)
   Winner 1: None                                   Winner 2: None
   PE: 2
       1.000 Err Factor       0.333 Desired
       0.333 Sum              0.333 Transfer            0.333 Output
*** 0 Weights            0.000 Error              0.000 Current Error
*** Repeat for PE's here on, 0 weights, 0 error.
   PE: 3
       1.000 Err Factor      -1.000 Desired
      -1.000 Sum             -1.000 Transfer           -1.000 Output
   PE: 4
       1.000 Err Factor      -0.273 Desired
      -0.273 Sum             -0.273 Transfer           -0.273 Output
   PE: 5
       1.000 Err Factor       0.231 Desired
       0.231 Sum              0.231 Transfer            0.231 Output
Layer: Hidden1
        PEs: 25        Wgt Fields: 2                      Sum: Sum
    Spacing: 5          F' offset: 0.00              Transfer: TanH
```

```
    Shape: Square                                              Output: Direct
   Scale: 1.00      Low Limit: -9999.00      Error Func: standard
 Offset: 0.00    High Limit: 9999.00               Learn: Delta-Rule
  Init Low: -0.100   Init High: 0.100      L/R Schedule: hidden1
   Winner 1: None                                      Winner 2: None
L/R Schedule: hidden1
   Recall Step                 1        0        0        0        0
     Firing Density     100.0000   0.0000   0.0000   0.0000   0.0000
     Gain                 1.0000   0.0000   0.0000   0.0000   0.0000
     Gain                 1.0000   0.0000   0.0000   0.0000   0.0000
   Learn Step              10000    30000    70000   150000   310000
     Coefficient 1        0.3000   0.1500   0.0375   0.0023   0.0000
     Coefficient 2        0.3000   0.1500   0.0375   0.0023   0.0000
     Coefficient 3        0.1000   0.1000   0.1000   0.1000   0.1000
   PE: 6
       1.000 Err Factor       0.000 Desired
       1.734 Sum              0.940 Transfer          0.940 Output
   ***  5 Weights         -0.000 Error          -0.000 Current Error
*** Repeat for PE's from here on, 5 weights, nearly 0 error.
   PE: 7
       1.000 Err Factor       0.000 Desired
      -2.111 Sum             -0.971 Transfer         -0.971 Output
   PE: 8
       1.000 Err Factor       0.000 Desired
      -0.297 Sum             -0.289 Transfer         -0.289 Output
   PE: 9
       1.000 Err Factor       0.000 Desired
       0.912 Sum              0.722 Transfer          0.722 Output
   PE: 10
       1.000 Err Factor       0.000 Desired
      -0.258 Sum             -0.252 Transfer         -0.252 Output
   PE: 11
       1.000 Err Factor       0.000 Desired
      -0.159 Sum             -0.158 Transfer         -0.158 Output
   PE: 12
       1.000 Err Factor       0.000 Desired
       0.169 Sum              0.168 Transfer          0.168 Output
   PE: 13
       1.000 Err Factor       0.000 Desired
      -0.342 Sum             -0.330 Transfer         -0.330 Output
   PE: 14
       1.000 Err Factor       0.000 Desired
       0.677 Sum              0.589 Transfer          0.589 Output
   PE: 15
       1.000 Err Factor       0.000 Desired
      -1.055 Sum             -0.784 Transfer         -0.784 Output
   PE: 16
       1.000 Err Factor       0.000 Desired
      -0.215 Sum             -0.212 Transfer         -0.212 Output
   PE: 17
       1.000 Err Factor       0.000 Desired
       1.487 Sum              0.903 Transfer          0.903 Output
   PE: 18
       1.000 Err Factor       0.000 Desired
      -0.250 Sum             -0.245 Transfer         -0.245 Output
   PE: 19
```

```
      1.000 Err Factor       0.000 Desired
      0.158 Sum              0.156 Transfer            0.156 Output
   PE: 20
      1.000 Err Factor       0.000 Desired
      1.666 Sum              0.931 Transfer            0.931 Output
   PE: 21
      1.000 Err Factor       0.000 Desired
     -2.920 Sum             -0.994 Transfer           -0.994 Output
   PE: 22
      1.000 Err Factor       0.000 Desired
      0.136 Sum              0.135 Transfer            0.135 Output
   PE: 23
      1.000 Err Factor       0.000 Desired
      0.118 Sum              0.117 Transfer            0.117 Output
   PE: 24
      1.000 Err Factor       0.000 Desired
     -0.597 Sum             -0.535 Transfer           -0.535 Output
   PE: 25
      1.000 Err Factor       0.000 Desired
      0.154 Sum              0.153 Transfer            0.153 Output
   PE: 26
      1.000 Err Factor       0.000 Desired
      0.203 Sum              0.201 Transfer            0.201 Output
   PE: 27
      1.000 Err Factor       0.000 Desired
     -1.358 Sum             -0.876 Transfer           -0.876 Output
   PE: 28
      1.000 Err Factor       0.000 Desired
      0.508 Sum              0.468 Transfer            0.468 Output
   PE: 29
      1.000 Err Factor       0.000 Desired
     -1.887 Sum             -0.955 Transfer           -0.955 Output
   PE: 30
      1.000 Err Factor       0.000 Desired
      0.345 Sum              0.332 Transfer            0.332 Output
Layer: Hidden2
        PEs: 25      Wgt Fields: 2                      Sum: Sum
    Spacing: 5        F' offset: 0.00              Transfer: TanH
      Shape: Square                                  Output: Direct
    Scale: 1.00     Low Limit: -9999.00         Error Func: standard
 Offset: 0.00    High Limit: 9999.00                 Learn: Delta-Rule
  Init Low: -0.100   Init High: 0.100         L/R Schedule: hidden2
   Winner 1: None                                Winner 2: None
L/R Schedule: hidden2
   Recall Step                   1         0         0         0         0
     Firing Density     100.0000    0.0000    0.0000    0.0000    0.0000
     Gain                 1.0000    0.0000    0.0000    0.0000    0.0000
     Gain                 1.0000    0.0000    0.0000    0.0000    0.0000
   Learn Step              10000     30000     70000    150000    310000
     Coefficient 1        0.2500    0.1250    0.0313    0.0020    0.0000
     Coefficient 2        0.3000    0.1500    0.0375    0.0023    0.0000
     Coefficient 3        0.1000    0.1000    0.1000    0.1000    0.1000
   PE: 31
      1.000 Err Factor       0.000 Desired
     -4.909 Sum             -1.000 Transfer           -1.000 Output
***   26 Weights        -0.000 Error           -0.000 Current Error
```

```
*** Repeat for PE's here on, 26 weights, nearly 0 error.
   PE: 32
       1.000 Err Factor        0.000 Desired
      -1.085 Sum              -0.795 Transfer          -0.795 Output
   PE: 33
       1.000 Err Factor        0.000 Desired
       3.423 Sum               0.998 Transfer           0.998 Output
   PE: 34
       1.000 Err Factor        0.000 Desired
       3.539 Sum               0.998 Transfer           0.998 Output
   PE: 35
       1.000 Err Factor        0.000 Desired
       0.414 Sum               0.392 Transfer           0.392 Output
   PE: 36
       1.000 Err Factor        0.000 Desired
      -1.275 Sum              -0.855 Transfer          -0.855 Output
   PE: 37
       1.000 Err Factor        0.000 Desired
       1.820 Sum               0.949 Transfer           0.949 Output
   PE: 38
       1.000 Err Factor        0.000 Desired
       3.687 Sum               0.999 Transfer           0.999 Output
   PE: 39
       1.000 Err Factor        0.000 Desired
       1.271 Sum               0.854 Transfer           0.854 Output
   PE: 40
       1.000 Err Factor        0.000 Desired
      -0.379 Sum              -0.362 Transfer          -0.362 Output
   PE: 41
       1.000 Err Factor        0.000 Desired
       0.636 Sum               0.563 Transfer           0.563 Output
   PE: 42
       1.000 Err Factor        0.000 Desired
      -0.823 Sum              -0.677 Transfer          -0.677 Output
   PE: 43
       1.000 Err Factor        0.000 Desired
       0.619 Sum               0.550 Transfer           0.550 Output
   PE: 44
       1.000 Err Factor        0.000 Desired
      -1.500 Sum              -0.905 Transfer          -0.905 Output
   PE: 45
       1.000 Err Factor        0.000 Desired
       2.516 Sum               0.987 Transfer           0.987 Output
   PE: 46
       1.000 Err Factor        0.000 Desired
       1.206 Sum               0.836 Transfer           0.836 Output
   PE: 47
       1.000 Err Factor        0.000 Desired
       0.972 Sum               0.750 Transfer           0.750 Output
   PE: 48
       1.000 Err Factor        0.000 Desired
       1.743 Sum               0.941 Transfer           0.941 Output
   PE: 49
       1.000 Err Factor        0.000 Desired
      -1.517 Sum              -0.908 Transfer          -0.908 Output
   PE: 50
```

```
        1.000 Err Factor       0.000 Desired
        0.166 Sum              0.165 Transfer          0.165 Output
    PE: 51
        1.000 Err Factor       0.000 Desired
        0.270 Sum              0.264 Transfer          0.264 Output
    PE: 52
        1.000 Err Factor       0.000 Desired
        0.125 Sum              0.124 Transfer          0.124 Output
    PE: 53
        1.000 Err Factor       0.000 Desired
       -1.336 Sum             -0.871 Transfer         -0.871 Output
    PE: 54
        1.000 Err Factor       0.000 Desired
       -0.958 Sum             -0.744 Transfer         -0.744 Output
    PE: 55
        1.000 Err Factor       0.000 Desired
        0.533 Sum              0.488 Transfer          0.488 Output
Layer: Hidden3
        PEs: 25       Wgt Fields: 2                        Sum: Sum
    Spacing: 5         F' offset: 0.00              Transfer: TanH
    Shape: Square                                   Output: Direct
  Scale: 1.00     Low Limit: -9999.00        Error Func: standard
Offset: 0.00    High Limit: 9999.00              Learn: Delta-Rule
Init Low: -0.100   Init High: 0.100        L/R Schedule: hidden3
   Winner 1: None                                Winner 2: None
L/R Schedule: hidden3
   Recall Step                 1         0         0         0         0
     Firing Density    100.0000    0.0000    0.0000    0.0000    0.0000
     Gain                1.0000    0.0000    0.0000    0.0000    0.0000
     Gain                1.0000    0.0000    0.0000    0.0000    0.0000
   Learn Step             10000     30000     70000    150000    310000
     Coefficient 1       0.2000    0.1000    0.0250    0.0016    0.0000
     Coefficient 2       0.3000    0.1500    0.0375    0.0023    0.0000
     Coefficient 3       0.1000    0.1000    0.1000    0.1000    0.1000
   PE: 56
        1.000 Err Factor       0.000 Desired
        0.824 Sum              0.677 Transfer          0.677 Output
***  26 Weights        -0.000 Error            -0.000 Current Error
***  Repeat for PE's here on, 26 weights, nearly 0 error.
   PE: 57
        1.000 Err Factor       0.000 Desired
        0.328 Sum              0.317 Transfer          0.317 Output
   PE: 58
        1.000 Err Factor       0.000 Desired
       -0.132 Sum             -0.131 Transfer         -0.131 Output
   PE: 59
        1.000 Err Factor       0.000 Desired
       -0.035 Sum             -0.035 Transfer         -0.035 Output
   PE: 60
        1.000 Err Factor       0.000 Desired
       -0.120 Sum             -0.120 Transfer         -0.120 Output
   PE: 61
        1.000 Err Factor       0.000 Desired
       -0.671 Sum             -0.586 Transfer         -0.586 Output
   PE: 62
        1.000 Err Factor       0.000 Desired
```

```
      -0.110 Sum              -0.110 Transfer          -0.110 Output
   PE: 63
       1.000 Err Factor        0.000 Desired
      -0.076 Sum              -0.076 Transfer          -0.076 Output
   PE: 64
       1.000 Err Factor        0.000 Desired
       0.697 Sum               0.602 Transfer           0.602 Output
   PE: 65
       1.000 Err Factor        0.000 Desired
      -0.083 Sum              -0.083 Transfer          -0.083 Output
   PE: 66
       1.000 Err Factor        0.000 Desired
      -0.117 Sum              -0.117 Transfer          -0.117 Output
   PE: 67
       1.000 Err Factor        0.000 Desired
      -2.059 Sum              -0.968 Transfer          -0.968 Output
   PE: 68
       1.000 Err Factor        0.000 Desired
       0.513 Sum               0.472 Transfer           0.472 Output
   PE: 69
       1.000 Err Factor        0.000 Desired
      -0.735 Sum              -0.626 Transfer          -0.626 Output
   PE: 70
       1.000 Err Factor        0.000 Desired
      -0.142 Sum              -0.141 Transfer          -0.141 Output
   PE: 71
       1.000 Err Factor        0.000 Desired
       0.405 Sum               0.384 Transfer           0.384 Output
   PE: 72
       1.000 Err Factor        0.000 Desired
       0.007 Sum               0.007 Transfer           0.007 Output
   PE: 73
       1.000 Err Factor        0.000 Desired
       3.931 Sum               0.999 Transfer           0.999 Output
   PE: 74
       1.000 Err Factor        0.000 Desired
       0.238 Sum               0.234 Transfer           0.234 Output
   PE: 75
       1.000 Err Factor        0.000 Desired
      -0.478 Sum              -0.444 Transfer          -0.444 Output
   PE: 76
       1.000 Err Factor        0.000 Desired
      -0.288 Sum              -0.280 Transfer          -0.280 Output
   PE: 77
       1.000 Err Factor        0.000 Desired
       0.474 Sum               0.441 Transfer           0.441 Output
   PE: 78
       1.000 Err Factor        0.000 Desired
      -8.096 Sum              -1.000 Transfer          -1.000 Output
   PE: 79
       1.000 Err Factor        0.000 Desired
       0.169 Sum               0.167 Transfer           0.167 Output
   PE: 80
       1.000 Err Factor        0.000 Desired
      -0.261 Sum              -0.255 Transfer          -0.255 Output
Layer: Out
```

```
       PEs: 1        Wgt Fields: 2                          Sum: Sum
   Spacing: 5         F' offset: 0.00                  Transfer: TanH
     Shape: Square                                       Output: Direct
   Scale: 1.00      Low Limit: -9999.00          Error Func: standard
Offset: 0.00    High Limit: 9999.00                  Learn: Delta-Rule
 Init Low: -0.100   Init High: 0.100           L/R Schedule: out
   Winner 1: None                                   Winner 2: None
L/R Schedule: out
   Recall Step                    1         0         0         0         0
     Firing Density      100.0000    0.0000    0.0000    0.0000    0.0000
     Gain                  1.0000    0.0000    0.0000    0.0000    0.0000
     Gain                  1.0000    0.0000    0.0000    0.0000    0.0000
   Learn Step               10000     30000     70000    150000    310000
     Coefficient 1         0.1500    0.0750    0.0188    0.0012    0.0000
     Coefficient 2         0.3000    0.1500    0.0375    0.0023    0.0000
     Coefficient 3         0.1000    0.1000    0.1000    0.1000    0.1000
   PE: 81
       1.000 Err Factor      -0.298 Desired
      -0.307 Sum             -0.298 Transfer             -0.298 Output
      26 Weights        0.000 Error                 0.000 Current Error
```

****

Decryption desired and actual output after convergence according to input of Table 4.1:

Desired:	Actual:
4780.000000	4779.549316
4942.000000	4941.904785
8523.000000	8523.464258
9880.000000	9880.255859
13751.000000	13750.194336
21386.000000	21385.947266
26946.000000	26945.638672
26139.000000	26138.501953
29325.000000	29324.567578
30022.000000	30022.140625
33050.000000	33049.261719
34609.000000	34609.441406
37175.000000	37174.546875
38939.000000	38939.292969
41305.000000	41305.357031
41525.000000	41525.300781
25907.000000	25907.408984
12828.000000	12828.163086
16986.000000	16985.839844
45926.000000	45925.791406
45353.000000	45353.366406
50536.000000	50535.578906
54086.000000	54086.265625
54097.000000	54097.269531
59988.000000	59988.027344
62803.000000	62803.003906
3018.000000	3017.567871
2445.000000	2444.980957

REFERENCES

1. W. F. Ehrsam, S. M. Matyas, C. H. Meyer, and W. L. Tuchman, "A Cryptographic Key Management Scheme for Implementing the Data Encryption Standard," IBM System Journal, Vol 17 No 2, 1978, pp.106-107.

2. Jennifer Seberry, Josef Pieprzyk, "Cryptography, An introduction to Computer Security," Advances in Computer Science Series, Australia Pty Ltd, 1989, pp 8–188.

3. W. S. Scott, R. N. Mayo, G. Hamachi, J. K. Ousterhous,"1986 VLSI Tools Still More Works by Original Artists," Report No. UCB/CSD 86/272, University of California at Berkeley, Berkely, CA 1986.

4. *VLSI Design Tools Reference Manual*, Northwest Laboratory for Integrated Systems, TR 88–09–01 University of Washington, Seattle, Sec 6.4 pp. 20–48.

5. Donald E. Knuth, *The Art of Computer Programming, Vol.2: Seminumerical Algorithm*, Addison Wesley, Reading MA, 1981, pp. 399–401, 419–420.

6. Carl Pomerance,"Cryptography and Computational Number Theory–An Introduction," Proceedings of Symposia in Applied Mathematics:American Mathematical Society, Vol 42, Rhode Island, 1990, pp.1–11.

7. D. H. Ackley, G. H. Hinton, T. J. Sejnowski, "A Learning Algorithm for Boltzman Machine," Cognitive Science 9, 1985, pp. 147–169.

8. G. W. Cottrell, P. Munro, D. Zipser, "Image Compression by Back–Propagation: An Example of Extensional Programming," ICS Report 8702, University of California at San Diego, Feb, 1987.

9. R. Hecht-Nielsen, *Neurocomputing*, Prentice Hall, Englewood Cliffs, NJ, 1991, pp. 2, 124–137.

10. Neuralcomputing, *Neuralworks Professional II/Plus and Neural Network Explorer*, Neuralware Inc., Pittsburg, 1991, pp. 3–11, 89–101.

11. S. Goldwasser, S. Micali, A. C. Yao,"On Signature and Authorization," Advances in Cryptology: Proceedings of Crypto '82, Plenum Press., 1983, pp. 211-215.

12. Giles Brassard, "Modern Cryptography," Lecture Notes in Computre Science, Vol 325, Springer–Verlag, NewYork, 1988, pp.20–31.

13. Kevin S. McCurley, "The Discrete Logarithm Problem", IBM Research Report RJ6877, Yorktown Heights, NY, 1989, pp20–21.

14. W. Diffie, M. G. Hellman,"New Direction in Cryptography," IEEE Transactions on Information Theory, Vol. IT–22, 1976, pp. 472–492, 644-654.

15. Ronald L. Rivest , "RSA Chips (Past/Present/Future)," Advances in Cryptology–Eurocrypt '84, Berlin 1985, p.159.

16. Carl Pomerance,"Factoring," Proceeding of Symposia in Applied Mathematics: American Mathematical Society Vol 42, RI, 1990, pp.27–48.

17. J. A. Davis, D. B. Holdridge, G. J. Simmons, "Status Report on Factoring," Advances in Cryptology– Eurocrypt '84, Berlin, 1985, p.183.

18. Carl Pomerance, "The Quadratic Sieve Factoring Algorithm," Advances in Cryptology– Eurocrypt '84, Berlin, 1985, pp.169–181.

19. P.A. Findley, B. A. Johnson, "Modulo Exponentiation Using Recursive Sums of Residues," Advances in Cryptology– Crypto '89, Springer–Verlag, pp. 371–386.

20. Neil West, Kamran Eshraghian, *Principles of CMOS VLSI Design*, Addison Wesley, Reading, MA, pp.212–224, 310–312.

21. Andrei Vladimirescu, Sally Liu,"The Simulation of MOS Integrated Circuits Using Spice2," Electronics Research Laboratory, UC Berkeley, UCB/ERL M80/7, October 1980.

22. Carver Mead, Lynn Conway, *Introduction to VLSI Systems*, Addison Wesley,Reading MA 1980, pp.150–153.

23. FIPS,"Data Encryption Standard", Federal Information Processing Standard Publication, 46–1, NBS, Jan, 1988.

24. R. Hecht-Nielsen, *Neurocomputing*, Prentice Hall, Englewood Cliffs, NJ, 1991, pp. 3–1, 89–101.

25. S. Luce, D. Martins, S. Nunn, J. Waters, "Exploring the Back-Propagation Model for Speech Application," Speech Tech', 1988, pp. 199–203.

26. *Neuralcomputing, Neuralworks Professional II/Plus and Neural Network Explorer*, Neuralware Inc., Pittsburg, 1991, pp. 3–11, 89–101.

27. Dorothy E. R. Denning, *Cryptography and Data Security* Addison Wesley, Reading MA 1982, pp.62–78.

28. John Wakerly, *Digital Design Principles and Practices*, Prentice Hall, Englewood Cliffs, NJ, pp. 484–486.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304–6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943–5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943–5002	1
4.	Dr. Chyan Yang, Code EC/Ya Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943–5000	2
5.	Dr. Herschel H. Loomis Jr, Code EC/Lo Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943–5000	1
6.	Director National Security Agency Attn: Mr Alan Chedester, V29 2800 Savage Rd Ft Meade, MD 20755	1
7.	LT Phong Nguyen 1236 Denton Avenue Hayward, CA 94545	2